RELIABILITY-YIELD ALLOCATION FOR SEMICONDUCTOR INTEGRATED CIRCUITS: MODELING AND OPTIMIZATION

A Dissertation

by

CHUNGHUN HA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2004

Major Subject: Industrial Engineering

RELIABILITY-YIELD ALLOCATION FOR SEMICONDUCTOR INTEGRATED CIRCUITS: MODELING AND OPTIMIZATION

A Dissertation

by

CHUNGHUN HA

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Way Kuo (Chair of Committee)	
Bryan L. Deuermeyer (Member)	Daniel W. Apley (Member)
Jianer Chen (Member)	Mark L. Spearman (Head of Department)

August 2004

Major Subject: Industrial Engineering

ABSTRACT

Reliability-Yield Allocation for Semiconductor Integrated Circuits:
Modeling and Optimization. (August 2004)
Chunghun Ha, B.A., Yonsei University;
M.S., Texas A&M University
Chair of Advisory Committee: Dr. Way Kuo

This research develops yield and reliability models for fault-tolerant semiconductor integrated circuits and develops optimization algorithms that can be directly applied to these models. Since defects cause failures in microelectronics systems, accurate yield and reliability models considering these defects as well as optimization techniques determining efficient defect-tolerant schemes are essential in semiconductor manufacturing and nanomanufacturing to ensure manufacturability and productivity. The defect-based yield model considers various types of failures, fault-tolerant schemes such as hierarchical redundancy and error correcting code, and burn-in effects, simultaneously. The reliability model counts on carry-over single-cell failures accompanied by the failure rate of the semiconductor integrated circuits under the assumption of an error correcting code policy. The redundancy allocation problem, which seeks to find an optimal allocation of redundancy that maximizes system reliability, is one of the representative problems in reliability optimization. The problem is typically formulated as a nonconvex integer nonlinear programming problem that is nonseparable and coherent. Two iterative heuristics, tree and scanning heuristics, and variants are studied to obtain local optima and a branch-and-bound algorithm is proposed to find the global optimum for redundancy allocation problems. The

proposed algorithms engage a multiple-search paths strategy to accelerate efficiency. Experimental results of these algorithms indicate that they are superior to the existing algorithms in terms of computation time and solution quality. An example of memory semiconductor integrated circuits is presented to show the applicability of both the yield and reliability models and the optimization algorithms to fault-tolerant semiconductor integrated circuits.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Professor Way Kuo, who provided guidance, encouragement in my research, and continuous financial support during this study. Special thanks go to my advisory committee Professor Bryan L. Deuermeyer, Professor Daniel A. Apley, and Professor Jianer Chen for their continuous attention. I have learned much from those individuals which was of value in completing this dissertation as well as of help in cultivating my career.

My fellow graduate students, Wen Luo, Jung Yoon Hwang, Linda Maria, Dini Sunardi, and Hsiang Lee deserve my thanks for their valuable comments and technical discussions on this research.

This dissertation is devoted to my parents, Jeong Kwang Ha and Jeong Ja Jeon. I could not have finished this research without their endless love, encouragement, and support. I am also indebted to my wife, Seong Hee Kim, for her sacrifice and support, and my daughter, Yeo Heun Ha, for the happiness and joy she brings to my life. I also want to thank to my parents-in-law for their continuous help to my family.

NOMENCLATURE

YIELDS

Y_{MFT}	manufacturing yield of a semiconductor integrated circuit
Y_{WP}	yield at wafer production stage
Y_{FAB}	yield at wafer fabrication stage
Y_{AP}	yield at assembly and packaging stage
Y_{BI}	yield at burn-in and final test stage
Y_{POI}	the Poisson yield model
Y_{NB}	the negative binomial yield model
Y_{YD}	yield related to yield defects
Y_{BD}	yield related to burn-in defects
Y_{IC}	yield of a semiconductor integrated circuit
Y_{PC}	yield of peripheral circuits
Y_{MB}	yield of a memory block
Y_{BF}	yield related to block failures
Y_{SC}	yield related to single cell failures
Y_{BI}	burin-in yield
Y_{DC}	yield of supporting circuits in a memory block
Y_{MA}	yield of memory segments in a memory block
Y_{LF}	yield related to line failures in a memory block

Y_{SCN} yield of a single cell without an ECC

Y_{BDC} yield of supporting circuits at a block redundancy

Y_{BLF} yield related to line failures in a block redundancy

Y_{BM} yield of a block module

Y_{BR} yield of a block redundancy

Y_{ICN} yield of a integrated circuit without fault-tolerance

RELIABILITIES

R_{IC} reliability of a integrated circuit in an useful life period

R_{NC} reliability of non-memory components in a chip

R_{SC} reliability of memory segments in a chip

AREAS

A_{chip} total area of a chip

A_{IC} total area of IC

A_{PC} area of peripheral circuits per HALF

A_{MB} area of a memory block

A_{MB0} area of a memory block without fault-tolerance

A_{BR} area of a block redundancy

A_{BR0} area of a block redundancy without fault-tolerance

A_{WL} area of a word line

A_{BL} area of a bit line

A_{BWL} area of a word line at a block redundancy

A_{BBL} area of a bit line at a block redundancy

A_c average critical area of all sizes of defects

A_Φ^c critical area for area Φ

FUNCTIONS

$R(t)$ reliability function at time t

$\mu(t)$ failure rate at time t

$h(t)$ hazard rate or instantaneous failure rate at time t

$D(x)$ defect density for defect size x

$A_c(x)$ critical area for defect size x

$f_s(x)$ probability density function for defect size x

$pof(x)$ probability of failure for defect size x

PARAMETERS

x defect size, diameter of a defect

x_0 the size of a defect with the highest probability of occurrence

s_0 the size of a critical defect

D_0 average defect density for all sizes of defects

λ the average number of defects (faults)

λ_i the average number of i type defects (faults)

α clustering factor for the negative binomial yield model

α_i clustering factor of i type failures

γ a ratio constant for a burn-in yield model

μ_{NC} failure rate of non-memory components in a chip

μ_{SC} failure rate of memory segments in a chip

t_0 mission time of an integrated circuit

N_{mb} the number of memory blocks

N_{wlc} the total number of memory words at a chip

N_{bm} the number of block modules with the same size of block redundancy

N_{wl} the number of word lines at a memory block

N_{bl} the number of bit lines at a memory block

N_{tbw} the total number of bits in in a memory word

N_{cbw} the number of correctable bits in a memory word

N_{par} the number of parity bits in a memory word

N_{bwl} the number of word lines at a block redundancy

N_{bbl} the number of bit lines at a block redundancy

VARIABLES

n_{br} the number of block redundancies

n_{wl} the number of row redundancies

n_{bl} the number of column redundancies

n_{bwl} the number of row redundancies at a block redundancy

n_{bbl} the number of column redundancies at a block redundancy

n_{ecc} existence of ECC

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Nanotechnology is considered one of the promising technologies for 21st century. The prefix "nano-" indicates one billionth (10^{-9}), i.e., one nanometer (nm) is equal to $10^{-9}m$ or $0.001\mu m$. In a narrow sense, *nanotechnology* means the design, fabrication, and manipulation of atoms and molecules. Therefore, it is sometimes referred to as *molecular technology*. In a broad sense, nanotechnology can be defined as any science and technology that manipulates matter and systems whose size or tolerance is less than $100nm$. It is widely known that classical Newtonian physics describes the interaction of matter with sizes of more than $100nm$ and that quantum physics handles matter less than $10nm$. Nanotechnology acts as a link between those two fields of physics.

Nanotechnology today is pervasive due to its broad applications. Anticipated applications include electronics, telecommunications, materials, pharmaceutical and medical fields, bioengineering, the environment, energy, space, and so on. It is generally accepted that human life will change dramatically when nanotechnology is successfully implemented in the real-world. Nanotechnology should also result in breakthroughs in other leading technologies. For example, evolution of information technology is required to manipulate large amounts of information in biotechnology, and the success of information technology will critically rely on nanotechnology in the form of nanocomputer and quantum computing.

Since the invention of Scanning Tunneling Microscopy (STM) in 1983 and

---

This dissertation follows the style of the *IEEE Transactions on Reliability*.

the discovery of the molecular structure, Fullerenes (C_{60}), in 1985, numerous research results have been announced by various institutes and academia related to nanotechnology over the last two decades. However, most of the results have not guaranteed feasibility in industry, but have been limited to the laboratory. To elevate nanotechnology from the science to the engineering, the assurance of manufacturability and productivity is indispensable. *Nanomanufacturing* is defined as a series of processes for building nano-systems whose scale is less than $100nm$. This includes various multidisciplinary areas such as materials, devices, design, architecture, fabrication, analysis and estimation, modeling and control, optimization, simulation, testing and inspection, various facilities for manufacturing, and so on.

The heart of nanomanufacturing is nanofabrication which is achieved by two fundamentally distinct approaches, *top-down* and *bottom-up*. The top-down approach follows a technology similar to current semiconductor manufacturing, such as photolithography, thin film diffusion, polymer molding, cutting, wrapping, polishing. Electron-beam lithography, molecular beam epitaxy, microcontact print, nanoimprint lithography, etc. are included in this approach. The main advantage of this approach is its repeatability which makes volume production possible. On the other hand, the bottom-up approach employs self-assembly methods using a molecular array and the stiction property of atoms through Scanning Probe Microscopes (SPM). Self-assembly, robotic nano-assembly, template develop, etc. are included in this class. Although the bottom-up approach is very difficult to develop, it is expected that bottom-up will be the dominate form of nanofabrication in the end because of the resolution limitations of the top-down approach.

During nanomanufacturing, large numbers of defects inevitably occur because of the tiny scale of the devices, complex manufacturing processes, contamination, insufficient resolution of the lithography facilities, uncertainty of the connectivity

between devices, and so on [1]. These defects reduce the yield and reliability of manufactured nano-systems, as a result, decrease manufacturability and productivity. It is impossible to detect all occurring defects, and is very expensive, exhaustive, and sometimes impossible to eliminate them. Nano-systems may also have severe reliability problems because they are very sensitive to external environmental factors such as temperature, humidity, and the electric field.

The ITRS Semiconductor Roadmap [2], assessed by several semiconductor associations at all over the world is the most reliable source for obtaining current and future physical and technical trends in semiconductor integrated circuits. According to the roadmap, the half pitch of manufactured semiconductor integrated circuits at 2002 is about $130nm$, and it will become $65nm$ by 2007 and $22nm$ by 2016 for dynamic random access memory (DRAM). In addition to the nano-scale, manufacturing technology of semiconductor integrated circuits is similar to the top-down approach of nanomanufacturing. Therefore, it is reasonable to include semiconductor manufacturing into the nanomanufacturing in a broad sense.

The assurance of manufacturing yield and system reliability is also very important in semiconductor manufacturing. The ITRS 2002 update [2] suggests several grand challenges related to yield and reliability. In the near future (through 2007), new reliability screen methods for burn-in, inspection techniques for non-visual defect sources with high aspect ratio inspection, and design methods considering manufacturing and test are considered the critical technologies. In the distant future (2008 through 2016), error-tolerant design techniques, yield models for new materials and devices, and integration of them will take an important role in semiconductor manufacturing. In summary, key technologies related to yield and reliability in nanomanufacturing as well as in future semiconductor manufacturing are defect detection and analysis, accurate yield modeling, and defect-tolerant design.

Critical defects cause failures in semiconductors. Since the manufacturing yield and reliability of manufactured systems rely on these failures, it is necessary to efficiently manage the defects. There are two approaches for enhancing yield and reliability on the basis of defects. The first approach is to decrease the number of defects, and the second approach is to develop defect-robust design. Decreasing the defects can be accomplished by tightly controlling the contamination level, by increasing the quality of the cleaning processes, and by precisely controlling the manufacturing processes using statistical process control methods. Defect-robust design can be achieved by designing physically reliable devices such as high-k dielectric gate materials and by employing fault-tolerant schemes. Since it is technically impossible to fabricate perfect devices and to eliminate all defects, defect-tolerance has been considered a worthy method for dealing with defects. Heath et al. [1] have defined defect-tolerance as the capability of a circuit to operate as desired without physical repair or the removal of random mistakes incorporated in the system during the manufacturing process. The most cost-efficient defect-tolerant method for improving yield and reliability is to assign and manage additional replaceable components in vulnerable components. These redundancies may have self testing and repairing ability, so called BIST (built-in self test) and BISR (built-in self repair) [3].

With any defect management methods, accurate yield and reliability models are essential. Accurate models are very useful for evaluating the feasibility of new devices, identifying and monitoring the sources of defects, providing accurate simulations for the improvement of physical and logical device designs, and so on. For the past four decades, various forms of yield and reliability models have been proposed [4, 5]. Yield models are primarily based on statistical inference for defect size distribution, defect density, critical area, and failure types. On the other hand, reliability models are primarily based on the physical behavior of the electronic de-

vices. There are many factors that affecting yield and reliability, such as the failure types, the system architecture, defect density, critical area, defect clustering, burn-in effects, fault-tolerant schemes such as redundancies and error correcting code, and so on. However, most of the models developed so far, only partially consider these factors due to the complex structure of, and the complicated correlation relationships of, these factors.

Since the late 1950's, redundancy techniques have been successfully employed and their effectiveness has been verified by diverse applications. Yield and reliability can be enhanced by adding redundancies on failed or unreliable components or by increasing the robustness of the components physically. Looking at the manner in which manufacturing and design technologies have developed, improving component reliability appears to have been generally preferred over adding redundancies in industry, because, in many cases, redundancy is difficult to add to real systems due to technical limitations and the relatively large quantities of resources, such as weight, volume, and cost, that are required. However, recently developed advanced technologies, such as semiconductor integrated circuits and nanotechnology, have revived the importance of the redundancy strategy [1, 6]. The current down-scaling trend in semiconductor manufacturing places certain limitations on enhancing reliability or yield by developing relevant physical technologies [7, 8]. Hence, various fault-tolerant and self-repairable techniques are generally recommended [6]. In fact, most advanced memory integrated circuits and VLSI, which include internal memory blocks, currently use a hierarchical redundancy scheme and an error correcting code.

The objectives of this dissertation are to develop accurate yield and reliability models for defect-tolerant semiconductor integrated circuits and to develop optimization algorithms that can be applied to various complex systems including the developed models; thus, the models and the optimization algorithms can be applied to

current semiconductor-systems and to future nano-systems with minor modifications. The defect-based yield and reliability models consider as many critical factors as possible for yield and reliability, which include various types of failures, burn-in effects, and fault-tolerant architecture with a hierarchical redundancy structure and an error correcting code, while the computational efficiency endures. Defect-tolerant architecture using redundancy increases yield and reliability, but consumes other resources such as area, volume, weight, cost, test and repair algorithm, and repair architecture. To efficiently achieve fault-tolerant systems, the number of redundancies should be optimized to maximize yield and reliability with the available resources, the so-called the redundancy allocation problem. Typical yield and reliability models are non-convex, nonlinear, nonseparable, and coherent, and optimization problems have the same properties. Coherent systems have component-wise increasing property, that is, adding redundancies increases the yield and reliability. In this study, I propose an efficient branch-and-bound approach for obtaining the global optimum and two iterative heuristics, a scanning heuristic and a tree heuristic, and their variants for achieving the local optimum of yield or reliability redundancy allocation problems. The global optimization method is based primarily on a search space elimination of disjoint sets in a solution space that does not require any relaxation of branched subproblems. The scanning heuristic finds a better local optimum by solving the problem from several systematically generated initial points, and the tree heuristic obtains several local optima by branching off solution paths from the main solution path, which is a set of points with a maximum sensitivity factor. The main advantage of these methods is flexibility (i.e., it does not rely on any assumptions of linearity, separability, single constraint, or convexity) which make the method adaptive to various applications.

This dissertation is organized into three major parts: modeling, optimization, and a case study. In Chapter II, the terms defect, fault, error, and failure, are defined

and classified. Then, the methods for managing defects are briefly reviewed. Chapter III begins with a description of yield and reliability. Basic defect-based yield models, the Poisson model and the negative binomial model, are reviewed considering defect size, defect density, and critical area. At the end, the defect-based burn-in yield models and the reliability model in an useful life period are introduced. In Chapter IV, an integrated yield model and a reliability model considering carry-over failures for memory integrated circuits are proposed based on the models in Chapter III.

Chapter V introduces coherent systems, and redundancy allocation problems are defined under mathematical programming techniques. In Chapter VI and VII, new algorithms for solving reliability optimization problems are proposed. Chapter VI offers a branch-and-bound algorithm employed for redundancy allocation problems. Chapter VII proposes two efficient iterative heuristics, the tree heuristic and the scanning heuristic, for redundancy allocation problems. Numerical experimentation in Chapter VI and VII shows that the proposed global and local optimization algorithms are superior to other existing algorithms for redundancy allocation problems in terms of computation time and/or solution quality.

In Chapter VIII, the optimization algorithms in Chapter VI and VII are applied to the integrated yield model and the reliability model in Chapter IV to obtain an optimal number of redundancies in an example problem. Finally, Chapter IX summarizes the work in the modeling and the optimization of semiconductor integrated circuits. The directions for future research in nanomanufacturing are described.

CHAPTER II

DEFECTS AND DEFECT MANAGEMENT

Semiconductor manufacturing consists of several stages of circuit design and mask preparation, wafer production, wafer fabrication, assembly and packaging, and burn-in and testing [9]. Each stage also includes a series of processes. For instance, wafer fabrication consists of oxidation, photoresist coating and etching, polysilicon masking and etching, ion implantation, metal deposition and etching, and so on. Figure 1 presents a diagram of the complex semiconductor manufacturing process. For modern complex semiconductor products, more than 400 steps of individual processes are required, and it often takes more than two months to complete the manufacturing process. The complexity of the semiconductor manufacturing process presents many opportunities defects to occur. Defects cause faults, errors, and failures in manufactured integrated circuits, and they result in yield loss and reliability degradation. In this chapter, defects and related terms are defined and classified, and defect management methods for enhancing yield and reliability are discussed.

II.1. Defect, Fault, Error, and Failure

Defect, fault, error, and failure are frequently used terms in semiconductor engineering as are yield and reliability engineering. In semiconductor engineering, a defect is defined as any physical imperfection which does not satisfy specified requirements. A fault is a critical defect which affects the performance or functional behavior of the integrated circuit. An error, which is a manifestation of a fault, is a discrepancy between the theoretical value of a correctly designed function and real observed value of the function in the semiconductor system. A failure is an event, or inoperable state,

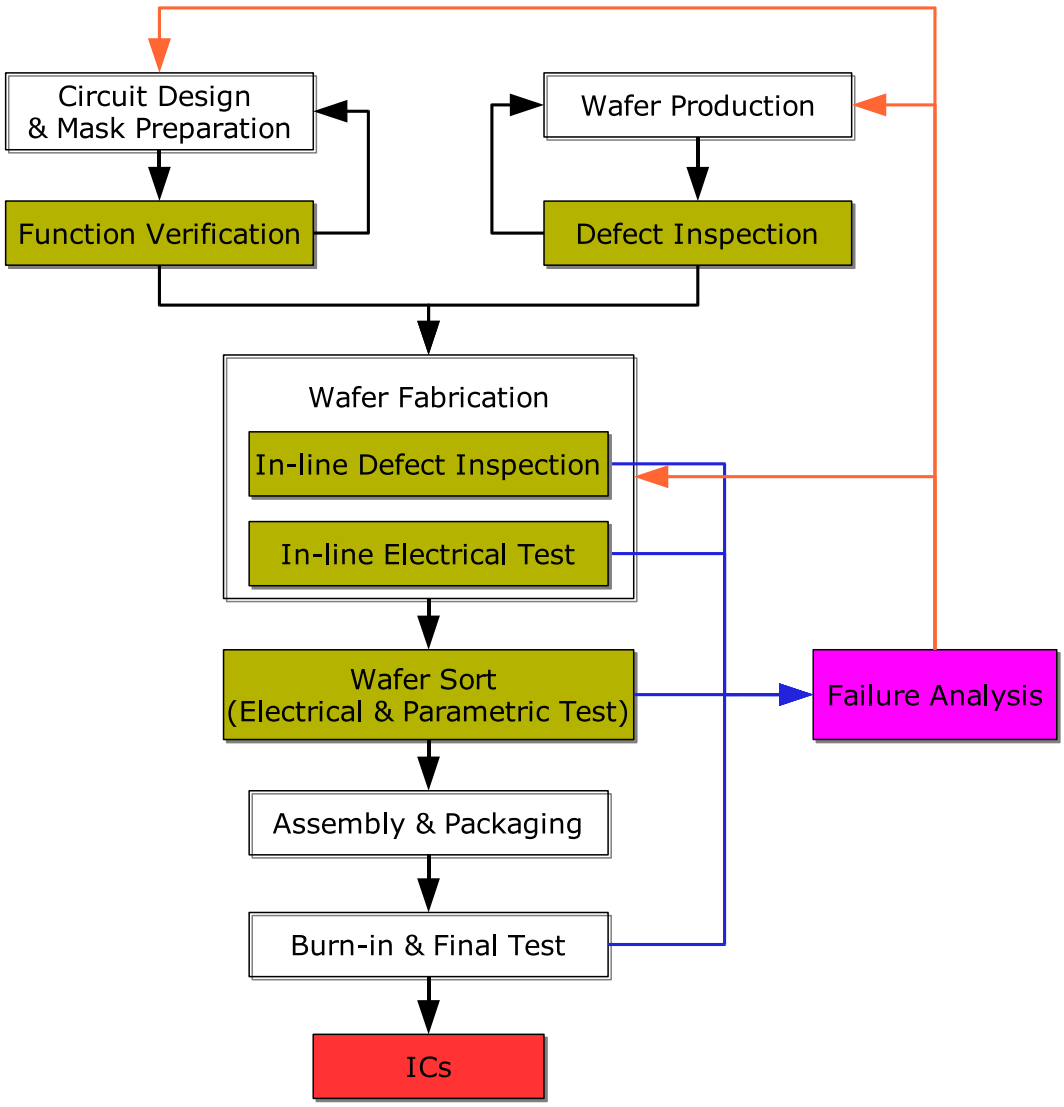

Figure 1: Manufacturing and Inspection Processes of Semiconductor Integrated Circuits

in which any item, or part of an item, does not, or would not, perform as previously specified [10]; in other words, failure is a status which does not function correctly without repair. In many cases, fault, error, and failure are used interchangeably to indicate a malfunction. However, strictly speaking, they have different definitions: a fault is physical and local; error is logical and functional; failure is physical and global. Not all defects result in faults, and not all errors are failures. For example, a memory bit can be temporarily changed by an α-particle in a memory integrated circuit and it can be fixed by an error correcting code immediately. In this case, the changed bit is not a failure, but an error.

To efficiently enhance yield and reliability, classification, diagnosis, and analysis of the defects and faults are very important. Since they are sources of failure, further investigation of them is required. Defects can be classified into several types according to their size, sources, location, and types as follows:

Global and point defects

Global defects, or gross area defects, are defects which occur on relatively large areas of a wafer surface. The main sources of global defects are wafer dislocations, wafer mishandling, mask misalignment, over etching or under etching, variation of implantation levels, and so on. Global defects can be gradually decreased during volume production by precisely controlling corresponding manufacturing processes. Point defects, or spot defects, are random local defects which mainly come from undesirable airborne particles, metallic impurities, and electrostatic discharge (ESD). They are mainly caused by cleanroom contamination, imperfect manufacturing processes, wafer mishandling, and so on. Since it is very difficult to control for spot defects, spot defects are a major concern of semiconductor manufacturers and, thus, a major focus of this research.

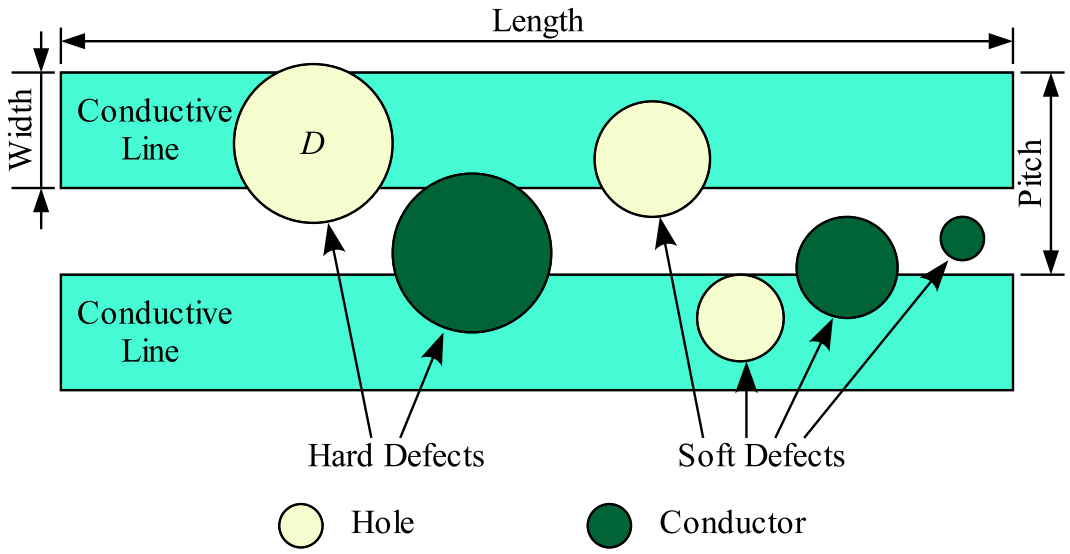

Figure 2: Hard and Soft Defects at an Intra-Layer

Hard and soft defects

Spot defects can be further classified into hard and soft defects. A hard defect is a spot defect which results in an open, or short, circuit; that is, failure of the integrated circuit. On the contrary, a soft defect does not affect the functionality of circuits, but it decreases or increases the physical dimensions of a device which may change the characteristics of the device. Soft defects can lead to system failure during manufacturing, the burn-in process, or operation. A graphical description of hard and soft defects appears in Figure 2. A defect is considered as a hard or soft defect based on size, location, and type. For example, the hard defect D in Figure 2 could be a soft defect if the defect was not a hole but a conductor.

Inter-layer and intra-layer defects

Based on their locations, spot defects are classified into inter- and intra-layer defects. Current complex integrated circuits have more than 8 layers built to avoid interconnections of the conductive lines which electronically connect devices [2]. Intra-layer defects, or photolithographic defects, are located on the same layer. Inter-layer de-

fects occur between two adjacent layers. Missing or extra material at the intra-layer or inter-layer may result in an open, or short, circuit, respectively.

Faults can be also classified into several types. The major types of faults are summarized below:

Functional and parametric faults

Functional faults, or logical faults, cause catastrophic behavior in a circuit device, which prevent it from performing its intended function correctly. Stuck-at-0 and stuck-at-1 faults, bridging, stuck-open or -short, and so on, are included in this category of faults. Parametric faults do not change any functional values, but they generate time delays of the signal or magnitude variations in some electrical parameters, such as voltage, current, and resistance, and capacitance.

Permanent and temporary faults

Permanent faults, or hard faults, are continuous and stable faults which endure regardless of time. Design faults, broken wire, and missing or extra material are permanent faults. On the other hand, temporary faults occur randomly (transient faults) or regularly with unknown intervals (intermittent faults). The transient faults are mainly caused by environmental conditions such as α-particle hits, cosmic rays, temperature variations, and electromagnetic interference. Intermittent faults are caused by non-environmental conditions such as variations of resistance and capacitance, noise, and wear-out.

Some types of defects and faults are correlated. They can be comprehend as a mixed form; for example, a soft spot defect at an intra-layer can be a parametric permanent fault. The possible combinations of the defects and faults are depicted as a block diagram in Figure 3.

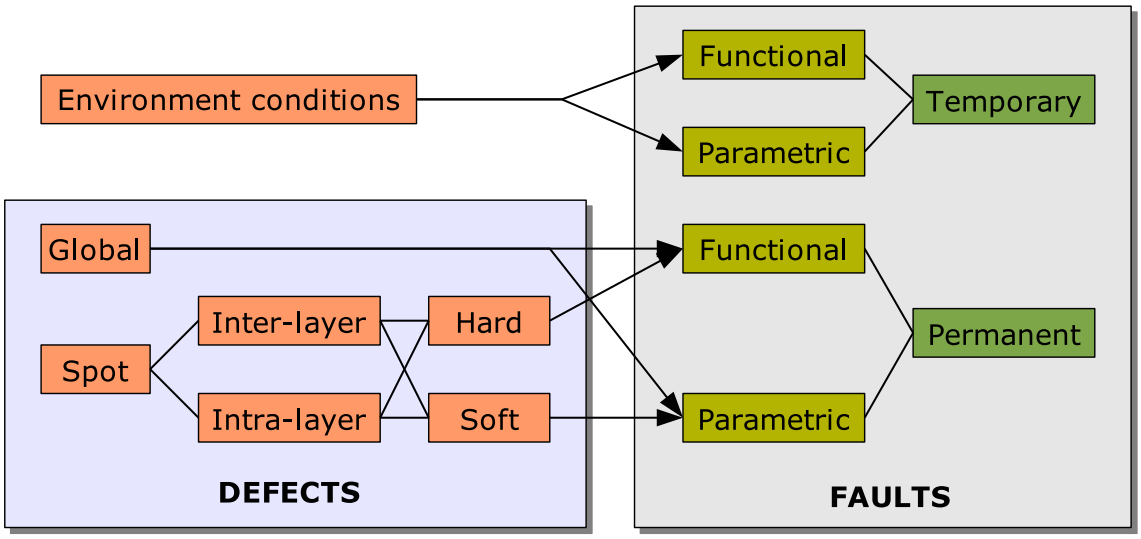

Figure 3: Possible Combinations of Defects and Faults

II.2. Defect Management

Various types of defects and faults can be detected by several different inspection processes. Defect inspection, or defect monitoring, can be performed during wafer fabrication (on-line inspections) using manual detection with an optical microscope and automatic detection with light scattering (or laser scattering) facilities. Global defects can be detected by a parameter monitoring approach, and local defects can be investigated by in-line monitors, gate-oxide monitors, and interconnect monitors. However, some defects may not be detected during any visual inspection process. At the end of wafer fabrication, a pattered wafer contains several hundred dies (or chips) which is a unit of the actual integrated circuits. The wafer sort process, probe testing or wafer probing, is performed to test parametric and electrical operation on each die by contacting bonding pads. Failed dies on the wafer sort process are marked, and discarded from further manufacturing stages. The various inspection processes and their flows during semiconductor manufacturing are depicted in Figure 1.

To enhance yield and reliability, manufacturing defects and faults should be managed effectively. One approach for defect management is to minimize the number of occurring defects and faults by controlling contamination levels and manufacturing processes with a low tolerance. The keys to this approach are how to retrieve main defect sources as correctly as possible to control the related processes, and how to perform the control process as precisely and fast as possible to increase yield as much, and as soon, as possible. Defect and fault classification, fault diagnosis, failure analysis, and statistical process control (SPC) are the main methods used to in this approach. However, there are certain limitations to this approach. Not all defects and faults can be detected by inspection processes because they may be located inside the physical layers or their size may be smaller than the resolution of the inspection facilities. Even when all of the defects and faults can be detected, it is still may not be possible to eliminate them entirely because of complex manufacturing processes, the resolution limit of the photolithography, or extremely high facilities costs.

Optical lithography is the key technology for future semiconductor manufacturing. Current extremely high density semiconductor integrated circuits are expected to meet its physical limitations in a few years [7, 8]. The most used light sources for lithography, at present, are the KrF laser (wavelength $248nm$) and the ArF laser (wavelength $193nm$). However, their resolution is not accurate enough for future use. The resolution of lithography is proportional to the wavelength of the light source and the processing techniques, and inversely proportional to the numerical aperture of the lens [9]. Due to technical difficulties, the resolution is generally similar to, or a little less than, the wavelength. Several alternative technologies, such as the F_2 excimer laser (wavelength $157nm$), extreme ultraviolet (wavelength $10 - 14nm$), and electron projection lithography, have been investigated by many research institutions for use in the near future [11]. However, as device dimensions decrease, the initial fa-

cility cost and the mask cost for lithography with the same or less resolution increase exponentially; and, as a consequence, so does the cost of fabrication of semiconductor integrated circuits.

Another approach to managing defects and faults is to design circuits and architecture that are more robust, the so-called defect-tolerant or fault-tolerant technique. This approach can be classified into two methods. The first method is to design devices on integrated circuits that are more tolerant to defects and other environmental variations. Most current semiconductor integrated circuits use CMOS (Complementary Metal Oxide Semiconductor) as a device. In current CMOS technology, the thickness of the gate dielectric (SiO_2) is less than 40Å [12] and the physical gate length is about $50nm$ [2]. Sizes continue to decrease to a few Å to meet the electrical requirements, such as capacity and resistance, of the new down-scaled devices. However, ultra-thin gate dielectric film invokes many manufacturing problems which affect yield and reliability. These problems include dielectric thickness variations, penetration of impurities from the gate into the gate dielectric, leakage current of the gate, and gate breakdown [12]. To overcome these problems, various high-k materials are being investigated as a substitute for the currently used SiO_2. With high-k materials, we can preserve the thickness of the gate dielectric, and, as a result, further down-scaling can be achieved without loss of yield and reliability. In addition to robust device design, yield can be also enhanced by decreasing the critical area of a chip. The critical area is the area where a certain size defect causes failure. The critical area can be decreased by optimizing the conducting lines, by optimizing the layout of devices, and by changing the floor plan of integrated circuits [6].

The second type of fault-tolerant techniques involves replacing failed devices with other working devices. Redundancy and error correcting codes (ECC) are the most used techniques in this category. From the late 1970's, these techniques have

been popular on memory integrated circuits; in fact, most advanced memory integrated circuits such as DRAM, SRAM, flash memories and microprocessors, which include an internal memory section, employ a hierarchical redundancy structure and ECC to increase yield and reliability [13]. There are several types of failures in the memory array related to the size and the location of defects. When a redundancy technique is employed on memory integrated circuits, word line failures, bit lines failures, cell failures, and block failures should be compensated for by the proper type of redundancies. At the end of wafer probing, this reconfiguration is typically performed by blowing and connecting built-in fuses/antifuses using laser equipments. However, the reconfiguration method using fuses has several disadvantages: 1) it requires additional processes which increase manufacturing cost; 2) it can not repair failures that occurred during operation due to its physical limitations; 3) it is unable to detect and test defects and faults; and 4) it is expensive due to the extremely high testing facility cost. Thus, the various automatic reconfiguration techniques such as BIST (built-in self test), BISD (built-in self diagnosis), and BISR (built-in self repair) have been proposed [3]. To perform the function correctly and efficiently, these built-in techniques require additional hierarchical cell array architecture and algorithms for finding the proper redundancies for detected failures.

CHAPTER III

YIELD AND RELIABILITY MODELS

Yield and reliability models can be classified into two types: physical models and statistical models. Physical models explain the electrical and parametric behavior of semiconductor devices. The famous Arrhenius equation for thermal behavior and the classic anode hole injection (1/E) model and the thermochemical (E) model for gate oxide are included in this class. This type of model is mainly used to improve reliability by changing the device characteristics. Statistical models are primarily based on statistical inference and the estimation of defect and failure data that are obtained in field. Most models, such as the Poisson and the negative binomial models for yield and the exponential and the Weibull models for reliability, belong to this category.

An accurate yield model is very useful for the following activities:

- verifying the productivity of current or new products by yield projection
- improving yield during volume production by diagnosing the defect types which cause most of the yield loss and monitoring relative manufacturing processes
- improving device design by providing accurate yields to simulation tools
- determining production control parameters by providing accurate estimations.

Similar to the yield model, reliability model is also very useful for the following activities [14]:

- evaluating the feasibility of new products
- comparing competing designs in terms of reliability

- identifying potential reliability problems
- planning maintenance and logistic support strategies
- providing input to other studies such as life-cycle cost analysis or product selection.

III.1. Manufacturing Yield

Manufacturing yield is defined as the ratio of the number of working units at the end of production to the total number of possible units at the beginning of production. Since semiconductor manufacturing consists of a series of stages, manufacturing yield can be calculated from the following equation:

$$Y_{MFT} = Y_{WP} \cdot Y_{FAB} \cdot Y_{AP} \cdot Y_{BI}, \tag{3.1}$$

where Y_{WP}, Y_{FAB}, Y_{AP}, and Y_{BI} denote the yields of wafer production, fabrication, assembly & packaging, and burn-in stages of Figure 1, respectively. Since this study does not deal with the wafer production and assembly & packaging stages, for simplicity, we can assume that $Y_{WP} = 1$ and $Y_{AP} = 1$.

Many semiconductor engineers regard manufacturing yield as fabrication yield because the wafer fabrication stage is the most important stage, and the wafer production and the assembly & packaging stage are sometimes performed at different companies that are not fabrication companies. However, from the reliability point of view, this perspective is not appropriate. Yield can be interpreted as reliability at time zero. Since reliability considers the lifetime of a device during operation, the lifetime should be calculated from the customer's viewpoint, i.e., operation. However, if we assume yield to be the fabrication yield, there is a time gap between yield

and reliability because of the assembly & packaging and burn-in stages. For this reason, I will distinguish fabrication yield from manufacturing yield. Therefore, fabrication yield and manufacturing yield can be calculated as follows (with field data in percentage unit):

$$Y_{FAB} = \frac{\text{Average number of good chips per wafer}}{\text{Total number of possible chips per wafer}} \times 100\%,$$

where a chip or a die denotes a piece of a silicon wafer that contains the complete device.

$$Y_{MFT} = \frac{\text{Average number of working ICs per wafer}}{\text{Total number of possible chips per wafer}} \times 100\%,$$

where IC indicates the final products that are ready to ship to the customer.

III.2. Reliability of Semiconductor Integrated Circuits

Reliability is one of the important factors determining the productivity of semiconductor integrated circuits. Reliability is defined as the probability that a device operates properly for a given period of time (a mission time) under designated operating conditions. Each device has a lifetime which is the length of time that the device works properly. Let T be a random variable for the lifetime of a integrated circuit. If the mission time of the integrated circuit is unspecified, the reliability of the integrated circuit becomes a real-value function for the mission time, i.e., the so-called reliability function. Note that the mission time is not a random variable. The reliability function, $R(t)$, which is the probability that lifetime T is greater than mission time t, can be formulated as follows:

$$R(t) = \Pr(T > t) = \int_t^\infty f(\theta)d\theta, \tag{3.2}$$

where $f(\theta)$ is the probability density function (pdf) of lifetime T with respect to operating time θ. An unreliability function which implies the probability of failure at time t is also defined as follows:

$$F(t) = \Pr(T \leq t) = \int_0^t f(\theta)d\theta = 1 - R(t). \tag{3.3}$$

The failure rate is the ratio of failures during a particular interval given that the product works properly by time t. Let the interval be $(t, t+\Delta t]$. Then, the failure rate is:

$$\mu(t) = \frac{\Pr(t < T \leq t + \Delta t)}{\Delta t \Pr(t < T)} = \frac{F(t+\Delta t) - F(t)}{\Delta t R(t)} = \frac{R(t) - R(t+\Delta t)}{\Delta t R(t)} \tag{3.4}$$

The instantaneous failure rate, or hazard rate $h(t)$, is the limit of the failure rate as Δt approaches zero, and it is expressed as:

$$h(t) = \lim_{\Delta t \to 0} \mu(t) = \lim_{\Delta t \to 0} \frac{F(t+\Delta t) - F(t)}{\Delta t R(t)} = \frac{dF(t)/dt}{R(t)} = \frac{f(t)}{R(t)} \tag{3.5}$$

Now, take the integral on both sides of Equation (3.5). Then, the following mathematical relationship between $h(t)$ and $R(t)$ holds:

$$\int_0^t h(\theta)d\theta = \int_0^t \frac{1}{R(\theta)} \frac{dF(\theta)}{d\theta} d\theta = \int_0^t \frac{-1}{R(\theta)} dR(\theta) = -\ln R(\theta)|_0^t = -\ln R(t) + \ln R(0).$$

We can assume that $R(0) = 1$, i.e., no failure at time zero, because any product is totally reliable at the beginning of its life. Then, the reliability function at mission time t can be computed with a hazard rate as follows:

$$R(t) = \exp\left(-\int_0^t h(\theta)d\theta\right). \tag{3.6}$$

Assuming that the instantaneous failure rate $h(t)$ is a constant μ w.r.t. at time t, the

reliability function becomes:

$$R(t) = e^{-\mu t} \tag{3.7}$$

and it is a exponential distribution.

Failures are counted in calculating yield and reliability. In semiconductor engineering, failures can be classified into three types according to the failure source: extrinsic, intrinsic, and wear-out. If a failure is caused by unrevealed manufacturing defects, it is called an extrinsic failure. Defects which do not materialize into yield losses can be grown to failures during operation depending on the quantity of external and internal stresses. The extrinsic failures usually observed during the infant mortality period have a decreasing failure rate (DFR). These failures can be effectively screened by accelerated life testing and burn-in. Intrinsic failures, sometimes called defect-free failures, occur randomly during operation. These failures have a constant failure rate (CFR) and can be observed during the useful life period. The most important reliability measure in semiconductor integrated circuits, time-dependent dielectric breakdown (TDDB), is related to both extrinsic and intrinsic failures. Wear-out failures, which have an increasing failure rate (IFR), are the result of device aging such as electromigration and transistor degradation. In general, wear-out failures should be avoided in the design stage of semiconductor devices [15]. Thus, wear-out failures will be ignored in this study. Since actual semiconductor integrated circuits include all of the types of failure, the failure rate function of an integrated circuit can be represented by the famous bathtub curve by summing the three failure rate functions. Figure 4 describes all of the failure rate functions and their corresponding periods.

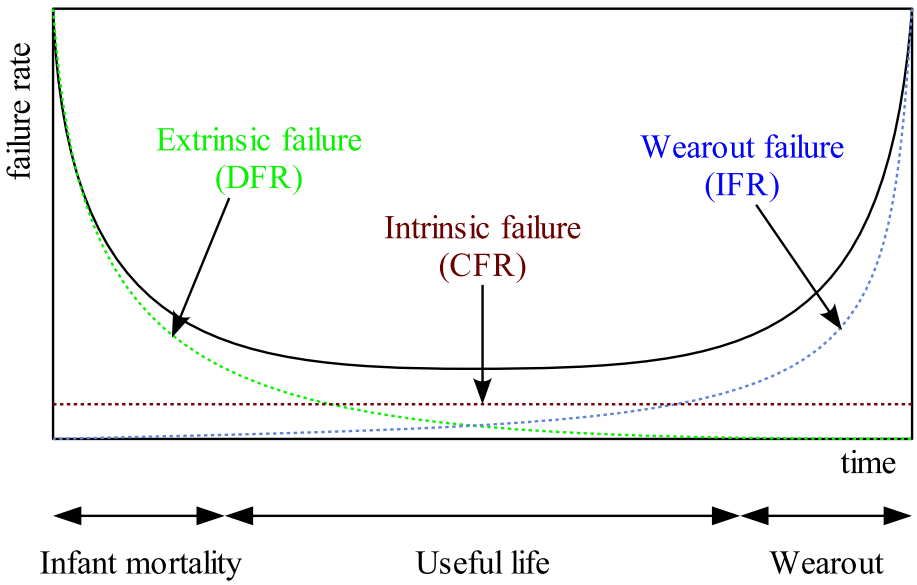

Figure 4: Bathtub Shape Failure Rate

III.3. Defect Size, Defect Density, and Critical Area

Defect-oriented yield models are significantly related to defect sizes, defect density, and critical area. Commonly, a spot defect, which is our major concern, is assumed to be a circle with a specific diameter, which is called the defect size. It has been verified empirically that the density of the number of defects has a peak at a specified defect size. The density linearly increases if the size of defect is less than the specific size and inverse polynomially decreases if the size of the defect is lager than the specific size. The defect size at the peak of the pdf depends on the contamination level and the resolution of the processes during manufacturing. Let the defect size be x and the specific size be x_0. The pdf of the defect size, $f_s(x)$, is generally described as follows [16]:

$$f_s(x) = \begin{cases} \dfrac{2(p-1)}{p+1}\dfrac{x}{(x_0)^2}, & 0 \le x \le x_0, \\ \dfrac{2(p-1)}{p+1}\dfrac{(x_0)^{p-1}}{x^p}, & x_0 \le x \le \infty. \end{cases} \tag{3.8}$$

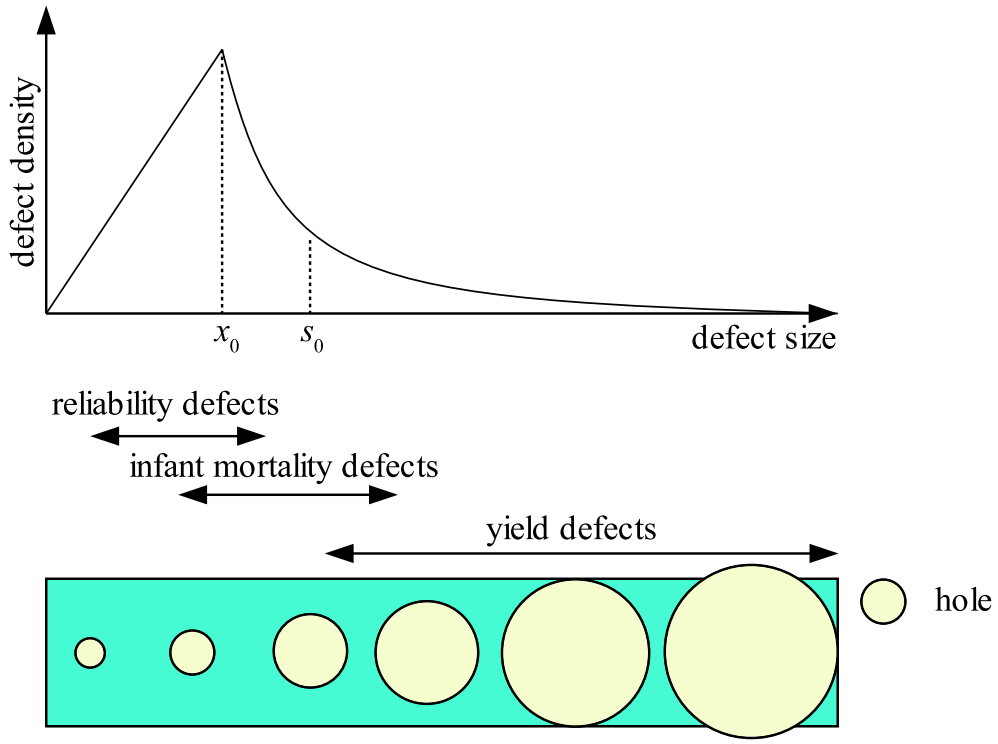

Figure 5: Distribution and Classification According to Defect Size

Many empirical data show that a reasonable value of p is in the interval [2.5, 3.7] [16]. The pdf for the defect size is described in Figure 5. In the figure, s_0 is the critical size of the defects which affect the yield, and a typical value of s_0 is a half of the minimum device feature size. Generally, x_0 is smaller than s_0.

Defect density, $D(x)$, is the number of defects of size x in a unit area (cm^2 or m^2). The average defect density for all defect sizes, D_0, can be computed by:

$$D_0 = \int_0^\infty D(x)dx.$$

Since $f_s(x)$ is a pdf of the defect size, the following relationship between the defect density and the pdf of the defect sizes holds:

$$D(x) = D_0 f_s(x).$$

The critical area is a set of locations where a defect with a certain size results in a failure of the integrated circuit. In other words, if a defect, which size is greater than, or equal to, x, falls in a critical area, $A_c(x)$, the integrated circuit fails. The probability of failure (pof) is the failure probability when a defect falls on a chip. Thus, the pof can be interpreted by a ratio of the critical area to the total area of the chip as follows:

$$pof(x) = \frac{A_c(x)}{A_{chip}}, \tag{3.9}$$

where A_{chip} is a total area of the chip. For all defect sizes, the average critical area can be computed by following:

$$A_c = \int_0^\infty A_c(x) f_s(x) dx,$$

In a yield model, the most important parameter is the average number of defects which is denoted by λ. λ can be computed with the following equation:

$$\begin{aligned} \lambda &= \int_0^\infty A_c(x) D(x) dx \\ &= D_0 \int_0^\infty A_c(x) f_s(x) dx \\ &= A_c D_0. \end{aligned} \tag{3.10}$$

To calculate the λ, we must know the critical area, $A_c(x)$ (or $pof(x)$ by Equation (3.9)), since D_0 can be obtained from the field data and $f_s(x)$ can be computed by Equation (3.8). There are various methods for achieving $A_c(x)$ and $pof(x)$. Geometrical methods applying a Boolean polygon operation can calculate $A_c(x)$ and Monte-Carlo methods can compute $pof(x)$. For a good summary and references on calculating the critical area, refer to [5]. These methods for calculating $A_c(x)$ or $pof(x)$ can be repeatedly performed at the design stage in order to decrease the

critical area of a chip.

III.4. Poisson and Negative Binomial Yield Models

The Poisson distribution is the most widely used discrete distribution for a statistical model with random occurrences in a specified interval or area. Since the number of defects varies from wafer to wafer and chip to chip, the Poisson model is appropriate for our purpose. To employ the Poisson distribution to a yield model, the following assumptions must be satisfied:

- all defects are chip-kill defects which induce failure of the integrated circuit
- all defects are mutually independent, i.e., the occurrence of a defect at any location does not affect the occurrence of any other defects.

Let a random variable X denote the number of defects in a chip. X follows the Poisson distribution with an intensity parameter of λ. The probability that a chip contains k defects is:

$$\Pr(X = k|\lambda) = \frac{e^{-\lambda}\lambda^k}{k!}, k = 0, 1, \ldots. \tag{3.11}$$

Since the mean of X is λ, λ can be interpreted as the average number of defects occurring on a chip - this coincides with the definition of λ in Section III.3. Since yield is the probability of a no chip-kill defect, the yield of the Poisson model can be formulated by Equation (3.10) and (3.11):

$$Y_{POI} = \Pr(X = 0|\lambda) = e^{-\lambda} = e^{-A_c D_0}. \tag{3.12}$$

Although the Poisson yield model is a reasonable yield model for the random occurrences of defects, the projected yield of the Poisson model is often pessimistic

because of the clustering effect of the defects. If defects are not evenly distributed on a wafer, some chips contain more defects than the average and some chips contain none. The failure of an integrated circuit does not depend on the number of defects but only on the existence of defects. Note that we are not yet employing defect-tolerant techniques. The clustering effect generally cause real yield to be larger than the projected yield using the Poisson model. To reflect the clustering effect of defects, many researchers have studied compound Poisson models [17, 18, 19].

The compound Poisson model assumes that the average number of defects, λ, (or the average defect density D_0) has a type of distribution. Murphy [17] compounded the Simpson distribution (triangular distribution) and the rectangular distribution, and Seed [18] applied an exponential distribution as a compounder. However, the most widely accepted compounder is the Gamma distribution [19]. Let L be a random variable which indicates the average number of defects. If L has a constant value A_cD_0, the pdf of L is a sifted Dirac δ function as follows:

$$f_L(\lambda) = \delta(\lambda - A_cD_0),$$

the yield can be computed by:

$$Y = \int_0^\infty e^{-\lambda} f_L(\lambda) d\lambda = e^{-A_cD_0}. \tag{3.13}$$

This coincides with the Poisson yield model of Equation (3.12).

Now, let the pdf of the random variable L be the Gamma distribution as follows:

$$f_L(\lambda|\alpha,\beta) = \frac{1}{\Gamma(\alpha)\beta^\alpha}\lambda^{\alpha-1}e^{-\lambda/\beta}, \;\; 0 < \lambda < \infty, \; \alpha > 0, \; \beta > 0,$$

where $EL = A_cD_0 = \lambda = \alpha\beta$ and $var(D) = \alpha\beta^2$. Since the number of defects on a chip, X, depends on the average number of defects, L, the random variable

X has a mixture distribution. It is well known that if X has a hierarchical mixture distribution of $X|L \sim \text{Poisson}(L)$ and $L \sim \text{Gamma}(\alpha, \beta)$ with integer α, the marginal distribution of X follows a negative binomial distribution. Thus, the probability that a chip contains k defects is:

$$\begin{aligned}
\Pr(X = k|\alpha, \beta) &= \int_0^\infty \frac{e^{-\lambda}\lambda^k}{k!} f_L(\lambda|\alpha, \beta) d\lambda \\
&= \frac{1}{k!\Gamma(\alpha)\beta^\alpha} \int_0^\infty \lambda^{\alpha+k-1} e^{-\lambda(1+1/\beta)} d\lambda \\
&= \frac{\Gamma(\alpha+k)\left(1+\frac{1}{\beta}\right)^{-(\alpha+k)}}{k!\Gamma(\alpha)\beta^\alpha} \int_0^\infty f_L(\lambda|\alpha+k, (1+1/\beta)^{-1}) d\lambda \\
&= \frac{\Gamma(\alpha+k)(\beta)^k}{k!\Gamma(\alpha)(1+\beta)^{\alpha+k}} \sim \text{negative binomial}\left(\alpha, \frac{1}{1+\beta}\right) \\
&= \frac{\Gamma(\alpha+k)\left(\frac{\lambda}{\alpha}\right)^k}{k!\Gamma(\alpha)\left(1+\frac{\lambda}{\alpha}\right)^{\alpha+k}} \qquad (3.14)
\end{aligned}$$

Since yield is the probability of a zero chip-kill defect, the yield can be computed by the following:

$$\begin{aligned}
Y_{NB} &= \Pr(X = 0|\alpha, \beta) \\
&= \left(1 + \frac{\lambda}{\alpha}\right)^{-\alpha} = \left(1 + \frac{A_c D_0}{\alpha}\right)^{-\alpha}. \qquad (3.15)
\end{aligned}$$

This yield model is normally called a generalized negative binomial yield model or a Stapper model [19]. In this model, the clustering factor, α, is used to adjust the clustering level compared to the observed field data.

Poisson and negative binomial yield models assume that the λ and the α are constants in the whole chip. However, in real applications, the chip area is often divided into several independent areas which have distinct values of the λ and the α. For example, a memory integrated circuits consists of several sections, including peripheral circuits and memory arrays. The critical area of the peripheral circuit area

is normally designed to be much smaller than that of the memory array area because any fault on the area cause chip failure. Thus, the λ of the peripheral circuit area is much less than the λ of the memory array area even though the defect density is the same. Consider a chip which consists of n statistically independent partitions. Let a random variable X_i be the number of defects for the ith section for $i = 1, \ldots, n$ and the total number of defects on the chip $X = \sum_{i=1}^{n} X_i$. Then, the probability that k defects occur at a chip is:

$$P(X = k) = \sum_{\mathbf{k} \in K} \prod_{i=0}^{n} P(X_i = k_i),$$

where k_i denotes the number of defects in the ith section, $\mathbf{k} = (k_1, \ldots, k_n)$, and $K = \{\mathbf{k} | \sum_{i=1}^{n} k_i = k\}$. Since yield is the probability of no defects on a chip, yield can be computed by:

$$Y = P(X = 0) = \prod_{i=0}^{n} P(X_i = 0) = \prod_{i=0}^{n} Y_i, \tag{3.16}$$

where Y_i is the yield of the ith section. If X_i follows the Poison distribution with parameter λ_i for $i = 1, \ldots, n$, the yield becomes:

$$Y = \prod_{i=0}^{n} P(X_i = 0 | \lambda_i) = \prod_{i=0}^{n} e^{-\lambda_i} = e^{-\overline{\lambda}}, \tag{3.17}$$

where $\overline{\lambda} = \sum_{i=1}^{n} \lambda_i$. If X_i follows the negative binomial distribution with parameters λ_i and α_i for $i = 1, \ldots, n$, the yield can be computed by:

$$Y = \prod_{i=0}^{n} P(X_i = 0 | \lambda_i, \alpha_i) = \prod_{i=0}^{n} \left(1 + \frac{\lambda_i}{\alpha_i}\right)^{-\alpha_i}. \tag{3.18}$$

III.5. Defect-based Burn-in Yield Models

Yield and reliability are two important factors for productivity and manufacturability, but they are different from each other. A simple way to divide them is to define yield as reliability at time zero. Time zero, however, is sometimes vague because of the burn-in process. Burn-in imposes excess electrical and thermic stresses to manufactured semiconductor products, which causes unreliable ones to fail early in the process. The failed products are discarded from the population. Since the remaining products have passed the infant mortality period, the reliability of the final products increase. Depending on the time zero, burn-in is included in the yield or the reliability calculation. Many semiconductor engineers include the burn-in process in the reliability area. However, as I mentioned before in Section III.1, I will include the burn-in process in yield area.

According to whether defects affect yield, burn-in, and reliability, they can be categorized into yield defects, burn-in defects, and reliability defects. Yield defects, or chip-kill defects, which have a relatively large size, cause failures at the wafer fabrication stage, so the defects are considered in calculating fabrication yield. Burn-in defects, or infant mortality defects, which typically are of medium size, do not cause failures at the fabrication stage, but result in failures in burn-in stage. Reliability defects, which have a relatively small size, do not cause failures during manufacturing, but they result in failures during operation. From now on, I will use the subscripts YD for yield defects and BD for burn-in defects to distinguish burn-in defects from yield defects. The types of failures are determined by the size and location of the defects; in other words, there are some interactions between the defects. Figure 5 describes the relationship between defect size and the classification of defects.

Since a defect can be the yield or the burn-in defects, many researchers have

attempted to find an unified model for both yield and burn-in defects, the so-called yield-reliability model [15, 20, 21, 22, 23]. The first model was introduced by Huston and Clarke [15]. Their model employs both a average critical area for yield defects, A_{YD}, and a average critical area for burn-in defects, A_{BD}, in the unified form. Similar to A_{YD}, A_{BD} is defined as a average critical area where the defects result in failures during the burn-in process. A_{BD} can be also obtained by a Monte-Carlo simulation. Assume that the number of defects follows the Poisson distribution. Then, yield can be computed by the following equation using Equation (3.12):

$$Y_{YD} = Y_{YD0} e^{-D_{YD} A_{YD}}, \tag{3.19}$$

where Y_{YD0} is the yield of the non-random yield defects and D_{YD} is the average defect density for the yield defects. Similar to the yield for yield defects in Equation (3.19), the yield for burn-in defects can be formulated with A_{BD}:

$$Y_{BD} = Y_{BD0} e^{-D_{BD} A_{BD}}, \tag{3.20}$$

where Y_{BD0} is the burn-in yield of the non-random burn-in defects, and D_{BD} is the average defect density for the burn-in defects. The D_{BD} is assumed to be the same as the D_{YD}. Thus, if we ignore the non-random terms Y_{YD0} and Y_{BI0}, the relationship between the yields for yield defects and burn-in defects can be formulated as following the equation by the simple manipulation of Equations (3.19) and (4.17):

$$Y_{BD} = (Y_{YD})^{A_{BD}/A_{YD}}. \tag{3.21}$$

Kuper and van der Pol [22, 23] proposed another yield-reliability model. They assumed that there is no difference between yield defects and burn-in defects but that only some of the yield defects become burn-in defects. This assumption implies that

the average defect density for burn-in defects is a fraction of the average defect density for yield defects, i.e, $D_{BD} = \alpha D_{YD}$, where α is a constant much less than 1. They employed the Poisson yield model as follows:

$$Y_{YD} = Me^{-D_{YD}A}, \tag{3.22}$$

where A is the total area, or the average critical area, of a chip and M is the maximum possible yield fraction which reflects the clustering effects. Similarly, the burn-in yield can be formulated by the Poisson model as follows:

$$Y_{BD} = e^{-D_{BD}A}. \tag{3.23}$$

Combining Equations (3.22) and (3.23), the following relationship holds:

$$Y_{BD} = \left(\frac{Y_{YD}}{M}\right)^{\alpha}. \tag{3.24}$$

The Kim and Kuo model [21] considered time-dependent dielectric breakdown which is one of the major failures in integrated circuits. Let Z_{OX} be the thickness of the gate oxide and Z_{EOX} is the effective oxide thickness, which is the remaining thickness of the gate oxide due to defects. Then, the following relationship holds for Z_{OX}, Z_{EOX}, and s:

$$Z_{EOX} = sZ_{OX}, \tag{3.25}$$

where s is the severity of the defect growth at time t during operation temperature T_0. The severity factor is calculated by the following equation[21]:

$$s = \frac{t}{\tau_0(T_0)} \exp\left[-\frac{G(T_0)Z_{OX}}{V_{OX}}\right],$$

where $\tau_0(T_0)$ and $G(T_0)$ are constants depending on temperature and V_{OX} is the

voltage across the gate oxide. Now, consider a single gate oxide where the critical area for yield defects is A_{YDOX} and critical area for burn-in defects is A_{BDOX}. Then, the ratio of critical area can be computed by following equation:

$$\frac{A_{BDOX}}{A_{YDOX}} = \frac{Z_{OX}^2}{(Z_{OX} - Z_{EOX})^2} - 1 = \frac{1}{(1-s)^2} - 1. \tag{3.26}$$

By incorporating Equation (3.26) into Equation (3.21), the burn-in yield at burn-in time t can be computed by:

$$Y_{BD}(t) = (Y_{YD})^{A_{BDOX}/A_{YDOX}} = (Y_{YD})^{1/(1-s)^2-1}. \tag{3.27}$$

Barnett [24] assumed that the average number of yield defects λ_{YD} and the average number of burn-in defects λ_{BD} have the following relationship:

$$\lambda_{YD} = \gamma \lambda_{BD}, \tag{3.28}$$

where γ is a constant of less than one. Let $YD(m)$ and $BD(n)$ denote the events of exactly m yield defects and exactly n burn-in defects, respectively. If we employ a negative binomial yield model and assume the defects are statistically independent, the total yield, i.e., the probability of no yield or burn-in defects is:

$$Y = \Pr\{YD(0), BD(0)\} = \left(1 + \frac{\lambda_{YD} + \lambda_{BD}}{\alpha}\right)^{-\alpha},$$

and the probability of no yield defects is:

$$Y_{YD} = \Pr\{YD(0)\} = \left(1 + \frac{\lambda_{YD}}{\alpha}\right)^{-\alpha}, \tag{3.29}$$

and the probability of no burn-in defects given no yield defects is:

$$Y_{BD} = \Pr\{BD(0)|YD(0)\} = \left(1 + \frac{\lambda_{BD}}{\alpha + \lambda_{YD}}\right)^{-\alpha}, \tag{3.30}$$

By merging Equations from (3.28) to (3.30), the burn-in yield can be formulated by:

$$Y_{BD} = \left[1 + \gamma \left(1 - (Y_{YD})^{1/\alpha}\right)\right]^{-\alpha}. \tag{3.31}$$

III.6. Reliability Model in an Useful Life Period

In the useful life period, it is very difficult to develop a generalized reliability model because of the complex failure mechanisms of integrated circuits. Since the failure rate in this period is generally assumed to be a constant, a simple exponential reliability model of Equation (3.7) is extensively used:

$$R(t) = e^{-\mu t},$$

where t is the mission time and μ is the failure rate of an integrated circuit. General measurement unit of the failure rate is failures per 10^9 operation hours (FIT). In typical, λ stands for the failure rate, but, in this dissertation, I will use μ for the failure rate to distinguish it from the average number of defects in the yield model. For this exponential reliability model, both mean time to failure (MTTF) and mean time between failures (MTBF) are $1/\mu$.

Assuming the exponential reliability model, the only remaining problem is to find the appropriate value of μ because the mission time t is determined by the customer's needs and the manufacturer's strategy. There are two approaches to estimate μ. The first approach is to use accelerated life testing (ALT) [4, 25]. Most microelectronics products have a very long lifetime with a very low failure rate. ALT imposes excessive electrical and environmental stresses, which are greater than the normal operating conditions, for a reasonable amount of time. The lifetime and failure rate of a product, therefore, can be estimated by extrapolating the ALT results into

normal operating conditions. A temperature test for chemical reactions, a voltage test for time-dependant dielectric breakdown (TDDB), a humidity test for moisture resistance, and a current density test for electromigration are the widely used ALT methods in semiconductor manufacturing [25].

For example, let us consider ALT for temperature stress and voltage stress. It is generally agreed that the famous Arrhenius equation explains well the relationship between a chemical reaction and temperature as follows:

$$R_r = k_0 \exp\left(-\frac{E_a}{kT}\right), \tag{3.32}$$

where R_r is the chemical reaction rate; k_0 is a constant; E_a is the activation energy in the eV unit; k is the Boltzmann's constant (8.617e-5 eV/K); and T is the temperature in the K unit. Let T_1 be the temperature of the ALT and T_2 be the temperature at the normal operating condition. Then, the lifetime of the system can be estimated by modifying Equation (3.32) as follows:

$$L_{T2} = L_{T1} \exp\left[-\frac{E_a}{k}\left(\frac{1}{T_1} - \frac{1}{T_2}\right)\right], \tag{3.33}$$

where L_{T1} and L_{T2} are measured or estimated lifetimes related to temperature T_1 and T_2, respectively. For voltage stress, the following relationship is typically used:

$$L_{V2} = L_{V1} \exp\left[-B(V_2 - V_1)\right], \tag{3.34}$$

where L_{V1} and L_{V2} are lifetimes related to applied voltage V_1 and V_2, respectively, and B is the voltage acceleration constant which depends on the oxide film. The lifetime of the integrated circuit can be computed by projecting the accelerated conditions, T_1 and V_1, into the normal condition, T_2 and V_2, respectively. If we consider both of

the tests, the failure rate μ can be estimated with a confidence level as follows:

$$\mu = \frac{\mathcal{X}^2_{1-\alpha}(2n+2)}{2 \cdot A_f \cdot t_0} \cdot 10^9(\text{FITS}), \tag{3.35}$$

where r is the number of failures; α is the confidence level for the $\mathcal{X}^2$ distribution; n is the number of samples; t_0 is the test time; and A_f is the acceleration coefficient, i.e., $(L_{T2}/L_{T1}) \cdot (L_{V2}/L_{V1})$.

A second approach is to use industry or military standards such as MIL-STD-217, Bellcore, JAP, etc. With these standards, many factors are involved in the estimation of μ: temperature, quality, voltage stress, environmental factors, etc. The only difference in these standards is that they consider different factors and formulae to calculate μ. To see detailed explanations of and references to the various standards for microelectronic devices, refer to a good summary paper [14]. In fact, some formulae in the standards include the ALT feature. However, the estimated μ using the standards is often different from the actual field data for recently developed semiconductor devices because they do not reflect emerging devices and manufacturing technology.

CHAPTER IV

MODELING OF FAULT-TOLERANT MEMORY INTEGRATED CIRCUITS

Memory integrated circuits such as DRAM (dynamic random access memory), SRAM (static random access memory), and flash memories have often employed fault-tolerant techniques to enhance manufacturing yield at a low additional cost. The importance of fault-tolerant techniques will increase even more as microprocessors and system-on-a-chips (SoC) demand more memory capability on their chips. Since the 1980's, various yield and reliability models for fault-tolerant memory integrated circuits have been developed [5, 6, 26, 27, 28, 29]. However, most of these models gave only partial consideration to failure types, redundancies, error correcting codes, and burn-in effects. In this section, these yield and reliability models are integrated into the defect-based yield model and the reliability model that consider both hierarchical redundancies and error correcting codes.

IV.1. System Architecture of a Typical Memory IC

A modern memory integrated circuit consists largely of peripheral circuits for the controller and the interface circuits and memory arrays for information data storage. Figure 6 shows the design architecture of a typical memory integrated circuit. We define a memory block as a memory array which includes column and row redundancies. Each memory block consists of a sey of memory segments, row and column redundancies, and supporting circuits such as a row decoder and stitching region, a column decoder and sense amplifier, buses, selection switches, and a fuse box for replacing failed memories as redundancies. Block redundancy (BR in Figure 6) may

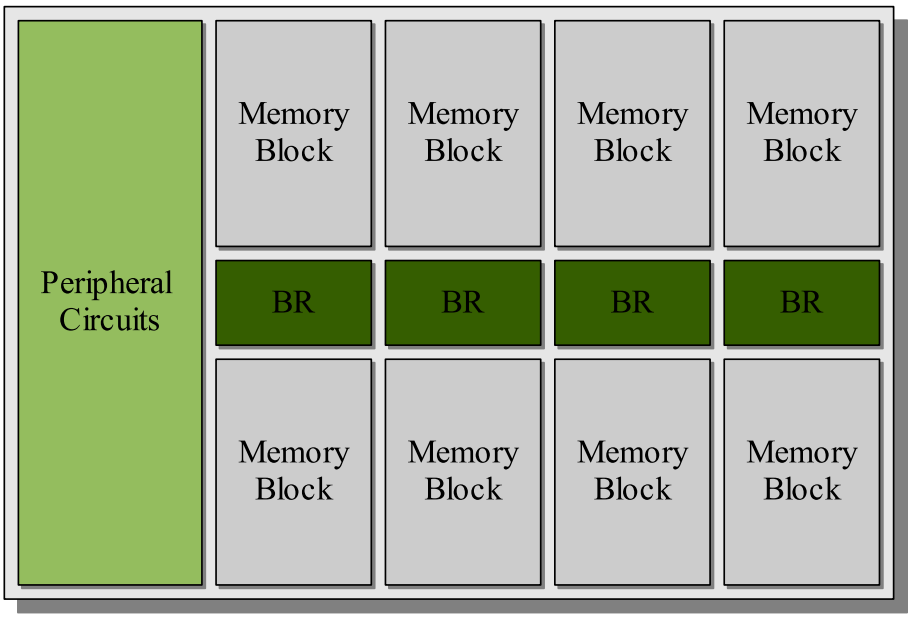

Figure 6: Schematic Diagram of a Typical Memory Integrated Circuit

also be added to integrated circuits to decrease the yield loss caused by block failures which often occur at the introductory stages of manufacturing [13]. A memory segment contains a number of memory words which consist of data bits and parity bits for error correction. Detailed memory chip architecture can be described as follows:

- total chip = peripheral circuits + memory blocks + block redundancies
- memory block = memory segments + row redundancies + column redundancies + supporting circuits
- block redundancy = memory segments + row redundancies + column redundancies + supporting circuits
- memory segment = memory words with error correcting code
- memory word = data bits + parity bits.

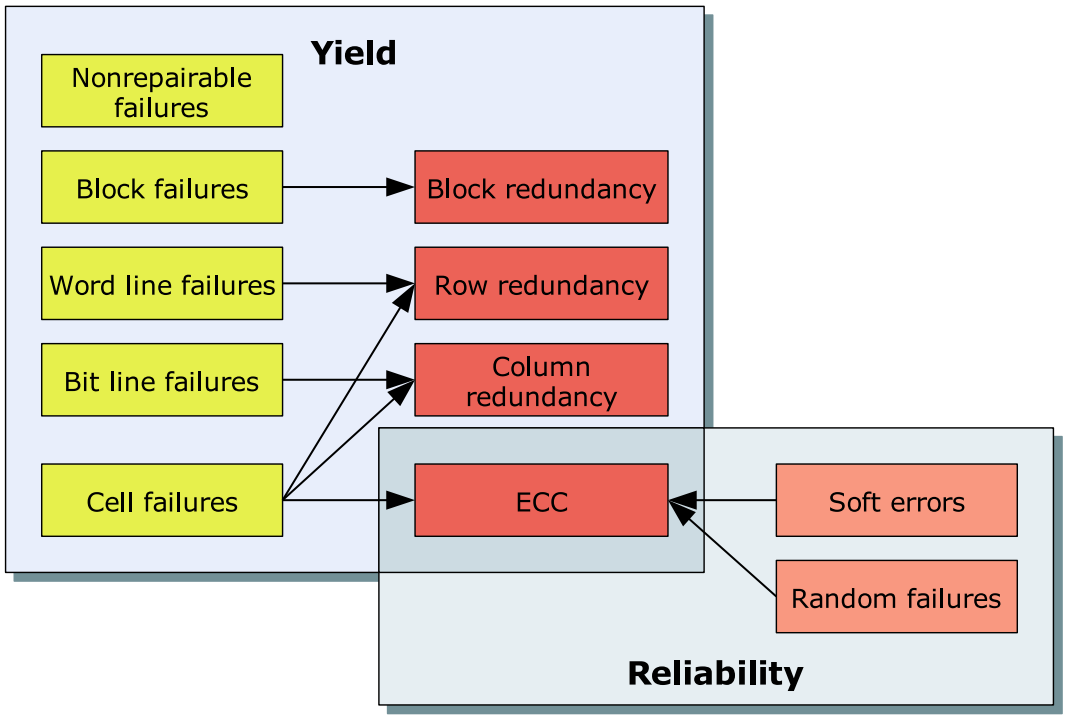

Figure 7: Relationship of Yield, Reliability, Failure Type, and Fault-Tolerant Scheme

Fault-tolerant schemes can be employed in any part of an semiconductor integrated circuit; however, we assume that fault-tolerance are applied for only memory arrays. In other words, failure is unavoidable if any faults fall in peripheral or supporting circuits. Faults occurring during wafer fabrication are covered by both redundancies and an error correcting code (ECC). On the other hand, faults occurring during operation are covered only by the ECC. Therefore, yield is affected by both redundancies and ECC and reliability is affected by only ECC. A row redundancy can repair any failed word line in the memory block and a column redundancy can replace any failed bit line in the memory block until corresponding redundancies are exhausted. A block redundancy can repair any failed memory segments with a size that is the same as the block redundancy in the whole memory array. The ECC can fix any failed bit in a word line.

Failures at the memory array can be classified into several types [29]: single cell failures (SC), single word line failures (SWL), two adjacent word lines failures (DWL), single bit line failures (SBL), two adjacent bit lines failures (DBL), and single cross lines failures (CL). A block failure (BK), or island failure, is separated from other failures because it can be repaired only by a block redundancy. Fault-tolerant schemes, i.e., row redundancy, column redundancy, block redundancy, and ECC, can repair several types of memory failures. For example, two row redundancies can repair two SWLs, one DWL, one SWL and one SC, or two SCs. The relationship between yield, reliability, failure type, and fault-tolerant scheme is described in Figure 7.

IV.2. Yield Model with Redundancy

Both redundancy and ECC are based on a redundancy technique. There are two types of redundancy: identical and nonidentical. With identical redundancy, the exact same memory module replaces a failed memory module. Row and column redundancy and ECC are included in this category. On the other hand, nonidentical redundancy is different. Block redundancy is categorized as nonidentical redundancy because block redundancy includes its own row and column redundancies, and it is different from a failed memory module in the memory array.

Consider a memory array which employs identical redundancy. In this case, redundancies tend to be surplus memory modules, the memory array is a k-out-of-n:G system. Let N_m be the required number of memory modules for the functioning of the system and n_r be the number of redundancies. The memory array system works properly if at least N_m modules function properly among the total of $(N_m + n_r)$ modules, i.e., N_m-out-of-$(N_m + n_r)$:G system. Then, the yield of the memory array

system is the probability of the survival of the N_m-out-of-$(N_m + n_r)$:G system as follows:

$$\begin{aligned} Y_{MA} &= \sum_{j=N_m}^{N_m+n_r} \binom{N_m+n_r}{j} (Y_M)^j (1-Y_M)^{N_m+n_r-j} \\ &= \sum_{j=0}^{n_r} \binom{N_m+n_r}{j} (Y_M)^{N_m+n_r-j} (1-Y_M)^j, \end{aligned} \tag{4.1}$$

where Y_M is the yield of a memory module. It can be the Poisson yield model of Equation (3.12) or the negative binomial model of Equation (3.15).

If the redundancy is nonidentical, the memory array system becomes a modified N_m-out-of-$(N_m + n_r)$:G system. Let Y_R be the yield of a redundancy which is different from the original memory module, i.e, $Y_R \neq Y_M$. The yield of the memory array system can be computed as follows:

$$\begin{aligned} Y_{MA} &= \Pr\{\text{at least } N_m \text{ modules work}\} \\ &= \sum_{j=0}^{n_r} \Pr\{\text{at least } j \text{ redundancies survive} | j \text{ memory modules fail}\} \\ &= \sum_{j=0}^{n_r} \left[\binom{N_m}{j} (Y_M)^{(N_m-j)} (1-Y_M)^j \sum_{k=j}^{n_r} \binom{n_r}{k} (Y_R)^k (1-Y_R)^{(n_r-k)} \right] \\ &= \sum_{j=0}^{n_r} \sum_{k=j}^{n_r} \binom{N_m}{j} \binom{n_r}{k} (Y_M)^{(N_m-j)} (1-Y_M)^j (Y_R)^k (1-Y_R)^{(n_r-k)}. \end{aligned} \tag{4.2}$$

IV.3. Yield Model with Various Types of Failures

For various types of memory failures, multivariate yield models have been developed [26, 29, 30]. Assume that SC, SWL, DWL, SBL, DBL, and CL occur independently. Let λ_i be the average number of the ith failure and x_i be the number of the ith failure, where i indicates a index for failure types, i.e., $I = \{SC, SWL, DWL, SBL, DBL, CL\}$. Let X_i be a random variable for the number of the ith failure. If row and column re-

dundancies are employed in the memory array, the Poisson yield model of the system for the various types of failures can be formulated as:

$$\begin{aligned} Y_{MA} &= \Pr\{\mathbf{x} \text{ is a fixable pattern}\} \\ &= \sum_{\mathbf{x}\in S} \prod_{i\in I} \Pr\{X_i = x_i\} \\ &= \sum_{\mathbf{x}\in S} \prod_{i\in I} \frac{\lambda_i^{x_i}}{x_i!} e^{-\lambda_i}, \end{aligned} \tag{4.3}$$

where $\mathbf{x} = (x_{SC}, x_{SWL}, x_{DWL}, x_{SBL}, x_{DBL}, x_{CL})$ and S is the set of fixable patterns which satisfies the following constraints [26]:

$$\begin{aligned} n_{wl} &\geq x_{SWL} + 2x_{DWL} + x_{CL}, \\ n_{bl} &\geq x_{SBL} + 2x_{DBL} + x_{CL}, \\ n_{wl} + n_{bl} &\geq x_{SC} + x_{SWL} + 2x_{DWL} + 2x_{CL} + x_{SBL} + 2x_{DBL}, \end{aligned} \tag{4.4}$$

where n_{wl} and n_{bl} denote the number of row redundancies and the number of column redundancies, respectively. Similar to the Poisson model, the negative binomial model with different clustering factors can also be developed as follows:

$$Y_{MA} = \sum_{\mathbf{x}\in S} \prod_{i\in I} \frac{\Gamma(\alpha_i + x_i)}{x_i!\Gamma(\alpha_i)} \frac{(\lambda_i/\alpha_i)^{x_i}}{(1+\lambda_i/\alpha_i)^{\alpha_i + x_i}}. \tag{4.5}$$

IV.4. Yield Model with ECC

ECCs have been developed to compensate for soft errors which occur during operation. An ECC can increase both yield and reliability by its auto-correcting characteristics. There are various ECC methods [3]. Among them, the DED/SEC (double error detect and single error correct) code, which uses odd weight Hamming code, is normally used in an ECC in memory integrated circuits. It is known that an optimal DED/SEC

consists of 128 data bits and 9 parity bits (total 137 bits) [3].

Stapper et al. [27, 28] have considered the effect of an DED/SEC on both yield and reliability models using the birthday problem under the assumption of the existence of single cell failures only. Let N_{wlc} be the total number of memory words in a chip. If we consider DED/SEC, only a failed bit can be repaired in a memory word. The probability of less than, or equal to, one failure in a memory word given k failures is:

$$\Pr\{\text{at most one failure in a memory word}|k \text{ failures}\} = \prod_{i=1}^{k} \frac{(N_{wlc} - k + 1)}{N_{wlc}}.$$

Let N_{SC} be a random variable which indicates the number of single cell failures and n_{sc} be the number of single cell failures in the memory array. If the negative binomial distribution of Equation (3.14) is applied, the yield of the memory array employing an DED/SEC using the negative binomial model can be derived as follows:

$$\begin{aligned}
Y_{ECC} &= \Pr\{\text{at most one failure in a memory word}\} \\
&= \sum_{n_{sc}=0}^{N_{wlc}} \Pr\{\text{at most one failure in a memory word, } N_{SC} = n_{sc}\} \\
&= \sum_{n_{sc}=0}^{N_{wlc}} \Pr\{\text{at most one failure in a memory word}|N_{SC} = n_{sc}\} \Pr\{N_{SC} = n_{sc}\} \\
&= \sum_{n_{sc}=0}^{N_{wlc}} \left\{ \left[\prod_{i=1}^{n_{sc}} \frac{(N_{wlc} + 1 - i)}{N_{wlc}} \right] \frac{\Gamma(\alpha + n_{sc}) \left(\frac{\lambda_{SC}}{\alpha}\right)^{n_{sc}}}{n_{sc}!\Gamma(\alpha) \left(1 + \frac{\lambda_{SC}}{\alpha}\right)^{\alpha + n_{sc}}} \right\}, \qquad (4.6)
\end{aligned}$$

where λ_{SC} is the average number of single cell failures per chip.

IV.5. Integrated Model for Memory Integrated Circuits

Throughout Section IV.2 to IV.4, various yield and reliability models have been investigated step by step for redundancy, various types of failures, and ECCs. There is

an obvious correlation among these models because a memory array can be expressed as a set of memory words, memory bits, memory segments, or memory cells at the same time. In addition, row redundancies, column redundancies, block redundancies, and ECCs can repair more than two types of failures (refer to Figure 7). Thus, to consider all of the factors simultaneously, special manipulations for the yield and the reliability models are required.

Since any failure in the peripheral circuits or memory block causes system failure, these are connected in series from the yield or reliability viewpoint. Block redundancies can repair any block failure in a memory array. Burn-in process is performed at the chip level. Therefore, a memory integrated circuit can be expressed as a series system of those components, and the total yield can be computed by:

$$Y_{IC} = Y_{PC} \cdot (Y_{MB})^{N_{mb}} \cdot Y_{BF} \cdot Y_{BI}, \tag{4.7}$$

where Y_{IC} is the yield of the manufactured integrated circuit; Y_{PC} is the yield of the peripheral circuits; Y_{MB} is the yield of a memory block; Y_{BF} is the yield related to block failures; Y_{BI} is the burn-in yield; and N_{mb} is the number of memory blocks.

In the following section, these yields and defect-based reliability are derived step by step.

Estimation of the average number of failures

The average number of failures, λ, is a key parameter for computing defect oriented yield and reliability models. λ can be obtained by multiplying the average critical area, A_c, and the average defect density, D_0. Data for the critical area is not specified for a part of a chip, but normally is provided based on the total chip area. Thus, to compute yields of a word line, a bit line, a cell, or a memory segment, which are used for calculating yield, some treatment is required. Let us assume that area A_j

for $j = 1, \ldots, k$ is a partition of the area A and all of the divided areas are identical, i.e., $A_i = A_j$ and $\lambda_i = \lambda_j$ for all i and j. Let $\overline{\lambda}$ be the average number of failures in area A. If we employ the Poisson model, the average number of failures in area A_j, λ_j, should be satisfied following the relationship expressed by Equation (3.17):

$$\overline{\lambda} = \sum_{j=0}^{k} \lambda_j = k\lambda_j \leftrightarrow \lambda_j = \frac{\overline{\lambda}}{k}.$$

Consider the negative binomial yield model. Let us assume that the clustering factor, α, is the same for the total area A and all of the partition A_j, and $\lambda_i = \lambda_j$ for all i and j. Then, the following relationship is true according to Equation (3.18):

$$\left(1 + \frac{\overline{\lambda}}{\alpha}\right)^{-\alpha} = \prod_{j=1}^{k} \left(1 + \frac{\lambda_j}{\alpha}\right)^{-\alpha} = \left[\left(1 + \frac{\lambda_j}{\alpha}\right)^{-\alpha}\right]^{k}.$$

Through some mathematical manipulation, the average number of failures in a partition A_j, λ_j, can be computed using the average number of failures in the total area A, $\overline{\lambda}$, as follows:

$$\lambda_j = \alpha \left[\left(1 + \frac{\overline{\lambda}}{\alpha}\right)^{1/k} - 1\right]. \tag{4.8}$$

Yield of peripheral circuits (Y_{PC})

Since we assume that peripheral circuits do not employ any fault-tolerant scheme, Y_{PC} can be computed simply by the Poisson yield model of Equation (3.12) or the negative binomial model of Equation (3.15). If we apply the negative binomial yield model, Y_{PC} is:

$$Y_{PC} = \left(1 + \frac{\lambda_{PC}}{\alpha_{PC}}\right)^{-\alpha_{PC}}, \tag{4.9}$$

where λ_{PC} and α_{PC} are the average number of defects and the clustering factor in the peripheral circuits, respectively.

Yield of a memory block (Y_{MB})

Now, consider a memory block. A memory block includes complex fault-tolerant architecture. Let us assume that failed memory modules are substituted for corresponding redundancies in the following order:

$$\text{word line} \rightarrow \text{bit line} \rightarrow \text{cell} \rightarrow \text{block}.$$

Since block failures are treated on the whole memory array, the effects for the block failures are excluded from calculating yield of the memory block. Then, yield of a memory block can be formulated as follows under the assumption of the independence of word line failures, bit line failures, and single cell failures:

$$\begin{aligned} Y_{MB} &= \Pr\{\text{supporting circuits work}\} \cdot \Pr\{\text{all of memory segments work}\} \\ &= Y_{DC} \cdot Y_{MA} \end{aligned} \tag{4.10}$$

where Y_{DC} is the yield of the supporting circuits in a memory block and Y_{MA} is the yield of all of memory segments in a memory block. Similar to Y_{PC}, Y_{DC} can be simply modeled as:

$$Y_{DC} = \left(1 + \frac{\lambda_{DC}}{\alpha_{DC}}\right)^{-\alpha_{DC}}, \tag{4.11}$$

where λ_{DC} and α_{DC} are the average number of defects and the clustering factor of the supporting circuits in a memory block, respectively.

Yield of a memory array, Y_{MA}, can be computed by Equation (4.5) using a set of fixable patterns. However, this method can not be applied directly to calculate Y_{MA} because single cell failures can be repaired by both remaining line redundancies and ECCs. Thus, we separate Y_{MA} again as $Y_{LF} \cdot Y_{SC}$, where Y_{LF} is a yield related to line failures and Y_{SC} is a yield related to single cell failures. Since Y_{LF} does not related to single cell failures, Y_{MA} can be calculated by Equation (4.5) after dropping

the last constraint, Equation (4.4).

Now, consider Y_{SC}. Let K be a random variable which indicates the number of the remaining row or column redundancies and X_l be a random variable which indicates the number of word or bit line failures. Let n_l be the total number of line redundancies, i.e., $n_{wl} + n_{bl}$, and λ_L be the average number of word line and bit line failures, i.e., $\lambda_{SWL} + 2\lambda_{DWL} + \lambda_{SBL} + 2\lambda_{DBL} + 2\lambda_{CL}$. Then, the probability of k number of the remaining line redundancies is:

$$\Pr\{K = k\} = \begin{cases} \Pr\{X_l \geq n_l | \lambda_L\} = 1 - \sum_{i=0}^{n_l - 1} \frac{\Gamma(\alpha + i) \left(\frac{\lambda_L}{\alpha}\right)^i}{i!\Gamma(\alpha) \left(1 + \frac{\lambda_L}{\alpha}\right)^{\alpha + i}}, & k = 0 \\ \Pr\{X_l = n_l - k | \lambda_L\} = \frac{\Gamma(\alpha + n_l - k) \left(\frac{\lambda_L}{\alpha}\right)^{(n_l - k)}}{(n_l - k)!\Gamma(\alpha) \left(1 + \frac{\lambda_L}{\alpha}\right)^{\alpha + (n_l - k)}}, & k \neq 0 \end{cases} \tag{4.12}$$

If an ECC is not employed, the yield for single cell failures is the probability that the number of single cell failures is less than or equal to the number of the remaining line redundancies given k number of the remaining line redundancies. Thus, the yield related to single cell failures without an ECC, Y_{SCN}, can be calculated by:

$$\begin{aligned} Y_{SCN} &= \sum_{k=0}^{n_l} \Pr\{X_{sc} \leq k | \lambda_{SC}\} \Pr\{K = k\} \\ &= \sum_{k=0}^{n_l} \left\{ \left[\sum_{i=0}^{k} \frac{\Gamma(\alpha_{SC} + i) \left(\frac{\lambda_{SC}}{\alpha_{SC}}\right)^i}{i!\Gamma(\alpha_{SC}) \left(1 + \frac{\lambda_{SC}}{\alpha_{SC}}\right)^{\alpha_{SC} + i}} \right] \Pr\{K = k\} \right\} \end{aligned} \tag{4.13}$$

If an ECC is employed, the yield related to single cell failures, Y_{SC}, can be calculated by Equation (4.6). However, in this case, the effect of the remaining line redundancies is ignored. Then, how do we calculate the λ_{SC} considering the remaining line redundancies? We have already derived Y_{SCN} in Equation (4.13), considering the effect of the remaining line redundancies. Thus, if we can extract λ_{SC} from Y_{SCN}, we can obtain λ_{SC} which reflects the effect of the remaining line redundancies. Through

some mathematical manipulation, we can easily obtain a new λ_{SC} as follows:

$$\lambda_{SC} = \alpha \left[(Y_{SCN})^{-1/\alpha} - 1 \right], \tag{4.14}$$

Yield of a block redundancy (Y_{BR})

A block redundancy has the same structure as a memory block, and thus the model is the same. The only difference is the consideration of the effect of an ECC. Memory words in a block redundancy also employ an ECC. However, it is very complex to calculate the yield of the memory array if we consider the effect of an ECC in the block redundancy. Block redundancies affect the yield of the memory array only when they replace failed block modules in the memory array. Thus, some single cell failures come from the block redundancies and the others come from the original memory array. This makes complex yield model of the memory array. This problem can be solved by not considering ECC on block redundancies, but by considering ECC on the memory array after the replacement of failed memory modules as block redundancies.

To distinguish from the memory block, I will attach B or b for the block redundancy at each subscript of models for the memory block. Then, the yield of a block redundancy can be formulated as follows:

$$Y_{BR} = Y_{BDC} \cdot Y_{BLF}, \tag{4.15}$$

where each yield can be formulated in the same way as the memory block.

Yield related to block failures (Y_{BF})

A block redundancy can cover any block module in the memory array. Let N_{bm} and n_{br} be the total number of block modules which have the same size of block redundancies and number of block redundancies in the chip, respectively. Since the

block redundancy also includes fault-tolerant architecture, the block redundancy is not identical to the block module. In this case, we can apply the yield model with nonidentical redunduncies of Equation (4.2). Then, Y_{BF} can be computed by:

$$Y_{BF} = \sum_{j=0}^{n_{br}} \sum_{k=j}^{n_{br}} \binom{N_{bm}}{j} \binom{n_{br}}{k} (Y_{BM})^{(N_{bm}-j)} (1-Y_{BM})^{j} (Y_{BR})^{k} (1-Y_{BR})^{(n_{br}-k)}, \quad (4.16)$$

where Y_{BM} is the yield of a block module and Y_{BR} is the yield of a block redundancy.

Burn-in yield (Y_{BI})

Burn-in yield can be formulated by Barnett's yield-reliability relationship of Equation (3.31):

$$Y_{BI} = \left\{ 1 + \gamma \left[1 - (Y_{ICN})^{1/\alpha_{ICN}} \right] \right\}^{-\alpha_{ICN}}, \quad (4.17)$$

where Y_{ICN} is the yield for yield defects. The actual number of yield defects decreases when some redundancies substitute for failed memory modules. However, the average number of yield defects is not changed because it depends on manufacturing processes. Thus, Y_{ICN} can be interpreted as yield without fault-tolerant schemes. Then, Y_{ICN} can be simply computed by Equation (3.18) as follows:

$$Y_{ICN} = \prod_{i \in \widetilde{I}} \left(1 + \frac{\lambda_i}{\alpha_i} \right)^{-\alpha_i}, \quad (4.18)$$

where $\widetilde{I} = \{PC, BK, SC, SWL, DWL, SBL, DBL, CL\}$.

Reliability in an useful life period

Detected failures are repaired by corresponding redundancies during the wafer probing process. Once this repair process is finished, no other repair jobs with redundancies can be performed during the operation unless BIST and BISR techniques are employed. Since a semiconductor integrated circuit is a series system of components,

the reliability of the integrated circuit can be represented as:

$$R_{IC} = R_{NC} \cdot R_{SC}, \tag{4.19}$$

where R_{NC} is the reliability of the non-memory components and R_{SC} is the reliability of memory segments related to reliability failures in a chip. R_{NC} and R_{SC} can be expressed by the exponential reliability model of Equation (3.7) in an useful life period:

$$R_{NC} = e^{-\mu_{NC} t_0}, \tag{4.20}$$

$$R_{SC} = e^{-\mu_{SC} t_0}, \tag{4.21}$$

where μ_{NC} and μ_{SC} are the failure rates of the non-memory components and memory arrays in a chip, respectively, and t_0 is the mission time of the integrated circuit.

If a number of single cell failures, which are not repaired by line redundancies, carry over to the useful life period, the reliability of the integrated circuit should be modified. Let N_{CF} be a random variable which denotes the number of carry-over single cell failures, N_{RF} be a random variable which indicates the number of reliability failures, and N_F be a random variable which denotes the number of total failures, i.e., $N_F = N_{CF} + N_{RF}$. Similar to Equation (4.6), the reliability of the integrated circuit with the DED/SEC can be formulated by:

$$\begin{aligned} R_{IC} &= R_{NC} \Pr\{\text{at most one failure in a memory word}\} \\ &= R_{NC} \sum_{n_f=0}^{N_{wlc}} \Pr\{\text{at most one failure in a memory word} | N_F = n_f\} \Pr\{N_f = n_f\} \\ &= R_{NC} \sum_{n_f=0}^{N_{wlc}} \left\{ \left[\prod_{i=1}^{n_f} \frac{(N_{wlc} + 1 - i)}{N_{wlc}} \right] \Pr\{N_f = n_f\} \right\}, \end{aligned} \tag{4.22}$$

where $\Pr\{N_f = n_f\}$ is the probability that the total number of failures is n_f. If we assume that the single cell failures and the reliability failures are independent, the probability can be computed by:

$$\begin{aligned}
\Pr\{N_f = n_f\} &= \Pr\{N_{CF} + N_{RF} = n_f\} \\
&= \sum_{n_{cf}=0}^{n_f} \Pr\{N_{CF} = n_{cf}\} \Pr\{N_{RF} = n_f - n_{cf}\} \\
&= \sum_{n_{cf}=0}^{n_f} \left\{ \frac{\Gamma(\alpha + n_{cf}) \left(\frac{\lambda_{SC}}{\alpha}\right)^{n_{cf}}}{n_{cf}!\Gamma(\alpha) \left(1 + \frac{\lambda_{SC}}{\alpha}\right)^{\alpha + n_{cf}}} \frac{\Gamma(\alpha + n_f - n_{cf}) \left(\frac{\mu_{SC}}{\alpha}\right)^{(n_f - n_{cf})}}{(n_f - n_{cf})!\Gamma(\alpha) \left(1 + \frac{\mu_{SC}}{\alpha}\right)^{\alpha + n_f - n_{cf}}} \right\}.
\end{aligned}$$

Note that we should use the new calculated λ_{SC} of Equation (4.14) for considering the carry-over single cell failures.

In this chapter, a large number of equations are derived for building the integrated yield model and the reliability model considering carry-over failures. To make them easier to understand, I have summarize these equations in Table 1.

Table 1: Formulas for Yield and Reliability Calculation

Yield	Formula	Equation
Y_{IC}	$[Y_{PC} \cdot (Y_{MB})^{N_{mb}} \cdot Y_{BF}] \cdot Y_{BI}$	(4.7)
Y_{PC}	$\left(1 + \frac{\lambda_{PC}}{\alpha_{PC}}\right)^{-\alpha_{PC}}$	(4.9)
Y_{MB}	$Y_{DC} \cdot Y_{LF} \cdot Y_{SC}$	(4.10)
Y_{DC}	$\left(1 + \frac{\lambda_{DC}}{\alpha_{DC}}\right)^{-\alpha_{DC}}$	(4.11)
Y_{LF}	$\sum_{\mathbf{x}\in S} \prod_{i\in I} \frac{\Gamma(\alpha_i + x_i)}{x_i!\Gamma(\alpha_i)} \frac{(\lambda_i/\alpha_i)^{x_i}}{(1+\lambda_i/\alpha_i)^{\alpha_i + x_i}},$	
	$I = \{SWL, DWL, SBL, DBL, CL\}$	(4.5)
Y_{SC}	$\sum_{n_{sc}=0}^{N_{wlc}} \left\{ \left[\prod_{i=1}^{n_{sc}} \frac{(N_{wlc}+1-i)}{N_{wlc}} \right] \frac{\Gamma(\alpha + n_{sc}) \left(\frac{\lambda_{SC}}{\alpha}\right)^{n_{sc}}}{n_{sc}!\Gamma(\alpha) \left(1 + \frac{\lambda_{SC}}{\alpha}\right)^{\alpha + n_{sc}}} \right\}$	(4.6)
Y_{BR}	$Y_{BDC} \cdot Y_{BLF}$	(4.15)
Y_{BDC}	$\left(1 + \frac{\lambda_{BDC}}{\alpha_{BDC}}\right)^{-\alpha_{BDC}}$	(4.11)
Y_{BLF}	$\sum_{\mathbf{x}\in S} \prod_{i\in I} \frac{\Gamma(\alpha_i + x_i)}{x_i!\Gamma(\alpha_i)} \frac{(\lambda_i/\alpha_i)^{x_i}}{(1+\lambda_i/\alpha_i)^{\alpha_i + x_i}},$	
	$I = \{SC, SWL, DWL, SBL, DBL, CL\}$	(4.5)
Y_{BF}	$\sum_{j=0}^{n_{br}} \sum_{k=j}^{n_{br}} \binom{N_{bm}}{j} \binom{n_{br}}{k} Y_{BM}^{(N_{bm}-j)} (1-Y_{BM})^j Y_{BR}^{(n_{br}-k)} (1-Y_{BR})^k$	(4.16)
Y_{BI}	$\left\{1 + \gamma \left[1 - (Y_{ICN})^{1/\alpha_{ICN}}\right]\right\}^{-\alpha_{ICN}}$	(4.17)
R_{IC}	$R_{NC} \sum_{n_f=0}^{N_{wlc}} \left\{ \left[\prod_{i=1}^{n_f} \frac{(N_{wlc}+1-i)}{N_{wlc}} \right] \Pr\{N_f = n_f\} \right\}$	(4.22)

CHAPTER V

OPTIMIZATION OF YIELD AND RELIABILITY

To efficiently constitute the fault-tolerant systems discussed in previous chapters, the number of redundancies and the size of the critical areas should be optimized. General yield and reliability models are nonconvex, nonlinear, nonseparable, and coherent. Depending on the decision variables of the problems, optimization problems can be either pure integer or mixed integer problems. In the following two chapters, newly proposed nonconvex integer nonlinear programming algorithms for coherent systems will be introduced. Tree and scanning heuristics find local optimum solutions and branch-and-bound method find global optimum solutions. Those methods can also be used to optimize general coherent systems.

V.1. Coherent System

Consider a reliability system where each component has two states. Let's assign a binary indicator variable x_i to the ith component, where x_i is one if the ith component is functioning and zero otherwise. The *system structure function* $\phi(\mathbf{x})$ can be defined if the binary state of the system is completely defined by the states of the components.

$$\phi(\mathbf{x}) = \begin{cases} 1 & \text{if the system is functioning} \\ 0 & \text{if the system is failing to function,} \end{cases} \tag{5.1}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ and n is the number of components in the system. Let $(\cdot, \mathbf{x}_i) = (x_1, \ldots, x_{i-1}, \cdot, x_{i+1}, \ldots, x_n)$ and $I = \{1, \ldots, n\}$. A *coherent* system can be defined as follows:

Definition 1. *A system is called coherent if, and only if, the structure function* $\phi(\mathbf{x})$ *satisfies the following conditions:*

i) (relevant) $\phi(0, \mathbf{x}_i) \neq \phi(1, \mathbf{x}_i)$ *for all* $i \in I$ *and for all* $(\cdot, \mathbf{x}_i)$,

ii) (increasing) $\phi(\mathbf{x})$ *is increasing and* $\phi(0, \mathbf{x}_i) \leq \phi(1, \mathbf{x}_i)$ *for all* $i \in I$ *and for all* $(\cdot, \mathbf{x}_i)$,

where the condition i) implies that all of the components are relevant to the system structure function and condition ii) indicates that the system structure function is nondecreasing. Let X_i be a random vector which denotes states of the ith component. All components are independent. Let $r_i = \Pr(X_i = 1)$. Since the system reliability is the probability that the system is functioning, the system reliability function is:

$$R_s(\mathbf{r}) = \Pr(\phi(\mathbf{X}) = 1) = E[\phi(\mathbf{X})],$$

where $\mathbf{X} = (X_1, \ldots, X_n)$ and $\mathbf{r} = (r_1, \ldots, r_n)$.

V.2. Formulation of Optimization Problems

The objective of the reliability optimization problems is to find the level of redundancy and/or reliability of the components which maximizes the system reliability under the given resource constraints, or minimizes the total cost under minimum system reliability and other resource limitations. Let's consider the following two reliability optimization problems:

[INLP]

$$\begin{aligned} \text{maximize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq b_i, \quad \text{for } i = 1, \cdots, m, \\ & l_j \leq x_j \leq u_j, \;\; x_j \in \mathcal{Z}_+, \quad \text{for } j = 1, \ldots, n, \end{aligned}$$

[**MINLP**]

$$
\begin{aligned}
\text{maximize} \quad & f(\mathbf{r}, \mathbf{x}) \\
\text{subject to} \quad & g_i(\mathbf{r}, \mathbf{x}) \leq b_i, \quad \text{for } i = 1, \cdots, m, \\
& l_j \leq x_j \leq u_j, \;\; x_j \in \mathcal{Z}_+, \quad \text{for } j = 1, \ldots, n, \\
& r_j \in (0, 1) \subset \mathcal{R}, \quad \text{for } j = 1, \ldots, n,
\end{aligned}
$$

where n and m are the number of components and the number of constraints, respectively; r_j is the component reliability of the jth component; x_j is the number of redundancies at the jth component; $\mathbf{x} = (x_1, \ldots, x_n)$; $\mathbf{r} = (r_1, \ldots, r_n)$; $f(\cdot)$ is an objective function of the problem; $g_i(\cdot)$ is the ith constraint function; b_i is the maximum allowable amount of the ith resource; $\mathcal{Z}_+$ and $\mathcal{R}$ denote a set of nonnegative integers and a real number, respectively. The objective function and constraint functions can be system reliability, manufacturing yield, manufacturing cost, profit, volume, area, etc. Redundancy can be added to components with various forms which include identical components, standby components, choices of multiple components, k-out-of-n systems, and others. Note that there are no restrictions of convexity, linearity, or separability on any of the objective functions or constraint functions.

If a system is coherent, the objective function $f(\cdot)$ and the constraint function $g_i(\cdot)$ are a monotonic increasing function over the feasible region $S = \{(\mathbf{r}, \mathbf{x}) | g_i(\mathbf{r}, \mathbf{x}) \leq b_i, r_j \in (0, 1), l_j \leq x_j \leq u_j, x_j \in \mathcal{Z}_+, \forall i, j\}$ and the solution space $\Omega = \{(\mathbf{r}, \mathbf{x}) | r_j \in (0, 1), l_j \leq x_j \leq u_j, x_j \in \mathcal{Z}_+, \forall j\}$. The problem is called separable if $f(\mathbf{r}, \mathbf{x}) = \sum_{j=1}^{n} f_j(r_j, x_j)$ and $g_i(\mathbf{r}, \mathbf{x}) = \sum_{j=1}^{n} g_{ij}(r_j, x_j)$, where $f_j(\cdot)$ is the jth subsystem reliability function and $g_{ij}(\cdot)$ is the consumption of the ith resource at the subsystem j.

There are various types of system configurations: *series*, *parallel*, *parallel-*

series, and *series-parallel*, *hierarchical series-parallel*, *complex*, etc. A series system works if all of the components work; a parallel system functions if any component works; a parallel-series system is a parallel system with series subsystems; a series-parallel system is a series system with parallel subsystems; a hierarchical series-parallel (HSP) system is a combined system with only series or parallel subsystems; a complex system is a system which can not be reduced into one component system using the parallel and series reduction technique. The system reliability function of a complex system can be computed by the pivotal decomposition method, the inclusion-exclusion method, or the sum-of-disjoint-products method [31]. The relationship of the components can be graphically expressed using a reliability block diagram. In the following chapters, system configurations are described by the reliability block diagram.

The overall system reliability depends on the type of system configuration. For example, the reliability of a series system and a parallel system can be computed by the following formulations, respectively:

$$\text{(series)} \quad R_s = \prod_{j=1}^{n} R_j,$$

$$\text{(parallel)} \quad R_s = 1 - \prod_{j=1}^{n} \left(1 - R_j\right),$$

where R_s is the overall system reliability; R_j is the jth subsystem reliability; and $\prod$ is the product operator. The system reliability function with a general system structure (including a complex or network structure) is nonlinear, nonconvex (or nonconcave), and nonseparable. Only series and parallel-series systems are known as quasi-concave (or log-concave) [32]. Since all reliability optimization problems include the system reliability function as the objective function or the constraint function, they become

nonconvex and nonlinear programming problems.

Reliability optimization problems are categorized into three typical problem types according to the nature of their decision variables: *reliability allocation*, *redundancy allocation*, and *reliability-redundancy allocation*. If the component reliabilities, r_j's for all j, are the only variables, the optimization problem is called a reliability allocation problem; if the number of redundancies, x_j's for all j, are the only variables, the problem becomes a redundancy allocation problem (**INLP**); if the decision variables of the problem include both component reliabilities and redundancies, the problem is called a reliability-redundancy allocation problem (**MINLP**). From the viewpoint of mathematical programming, the reliability allocation problem is a continuous nonlinear programming problem (NLP); the redundancy allocation problem is a pure integer nonlinear programming problem (INLP); and the reliability-redundancy allocation problem is a mixed integer nonlinear programming problem (MINLP). It is well known that all of the problems above are classified as NP-hard problems [33]. In other words, as yet, there is no dominant optimization algorithm for them which achieves a global optimal solution in polynomial time.

These problems are different from others due to several specific properties, which make them either harder or easier to solve. First, the general system reliability function is nonlinear, nonseparable, and nonconvex and so are the problems. Thus, various efficient integer and/or nonlinear programming techniques used to solve linear, separable, or convex problems are not directly employable. Second, the system under consideration is a coherent system which has component-wise increasing objective and constraint functions. This property provides good upper bounds for coherent functions in a specified solution space. Third, since there are various types of system structures, it is very difficult to develop an unified algorithm which is appropriate for all types of structures. Therefore, for global optimal solutions, enumeration methods

are used extensively. The proposed algorithms in Chapter VI and Chapter VII are developed to maximize the advantages and to minimize the disadvantages of the above properties.

V.3. Reliability Redundancy Optimization Algorithms

The redundancy allocation problem **INLP** is very difficult to solve. Chern [33] has proved that even the simplest redundancy allocation problems, a series system with one constraint or a series system with identical components and two constraints, are NP-hard. Due to their difficulty, various approaches, such as heuristics, approximations, and enumerations, have been considered for solving redundancy allocation problems. The characteristics of the three main types of algorithms are described below:

Heuristics

Heuristics find a local or a near optimal solution by gradually improving an incumbent solution using intuition. The steepest ascent method using sensitivity factors, boundary region search, increasing redundancy on minimal path sets, and meta-heuristics are in this category. Heuristics can be executed in a relatively short time and provide reasonably good solutions but the global optimality of the solution is generally not guaranteed.

Approximation methods

In approximations, a relaxed problem which has a larger feasible region than that of the original problem is solved and its optimal solution is rounded off to an integer solution. To solve the relaxed problem, mathematical programming techniques, such as linear, nonlinear, and geometric programming, are normally used. The main disadvantage of this type of algorithm is that it requires high quality information,

e.g., derivatives and mathematical reformulation; therefore, performance critically depends on the system structure.

Global optimization methods

Since the redundancy allocation problem is a nonconvex INLP, combinatorial optimization techniques can be used for global optimization methods. Dynamic programming [34], implicit enumeration [35], and branch-and-bound [34] are typical approaches in this category. Although the approaches guarantee global optimal solutions, they are very time consuming and their efficiency depends on the efficiency of the search space elimination.

Detailed descriptions of, and references for, the above three types of approaches are well summarized in [34, 36, 37].

CHAPTER VI

EXACT ALGORITHM FOR REDUNDANCY ALLOCATION

The most used global optimization method in redundancy allocation problems is dynamic programming (DP). The implementation of DP, however, is limited by the number of constraints and the system structures it can be applied to. For a system which has more than two constraints, the computational complexity of DP increases exponentially. Although this weakness can be compensated for by employing Lagrangian multipliers and dominating sequence techniques, DP is still not applicable to nonseparable systems such as reliability optimization problems with complex structures.

Contrary to DP, implicit enumeration and branch-and-bound methods do not depend on the separability of the objective and the number of constraint functions because they do not use recursive equations to reduce solution space. The early research on these methods was based on transforming integer decision variables into corresponding binary variables. However, the transformed problem, the so-called 0-1 programming problem, was often inefficient because a huge number of decision variables were generated if the range of variables was large [38].

The most efficient branch-and-bound method for redundancy allocation problems at present is the method developed by Nakagawa, et al. [39]. This method does not require a 0-1 programming transformation or relaxation of the integer variables. The most versatile relaxation these authors applied was dropping the constraints and fixing the value of the variables. This method evaluates the objective function only at the terminal nodes of the enumeration tree. Since the number of terminal nodes rapidly increases depending on the the number of variables and the integer interval of

each, the method is time consuming and inapplicable for solving large-scale problems.

Recently, Prasad and Kuo [35] proposed an implicit enumeration algorithm, the lexicographic search, which is an improved version of Misra's implicit enumeration method [38]. The algorithm searches every feasible solution lexicographically and finds the best one; during the search, infeasible solutions and solutions over a computed upper bound are eliminated. Although they only considered the problem with separable constraints in the paper [35], it can be easily implemented for a general integer programming problem in a coherent system. However, the algorithm is still exhausting if we do not use the associated solution space elimination techniques, which cannot be applied to nonseparable systems.

Due to their flexibility and their optimality properties, branch-and-bound methods have been extensively used in mathematical programming. The general branch-and-bound method for maximizing INLP problems is based on the following procedures: 1) find a feasible solution and set the current optimum as the value of the initial feasible solution; 2) if there is no unsolved subproblem, terminate the procedure; 3) branch into subproblems where a decision variable is fixed or bounded; 4) apply relaxation for each subproblem and solve the problem, and then update the upper bound as the maximum value of the solution; 5) fathom this tree node if the upper bound is less than the current optimum; 6) if the upper bound is feasible, update the current optimal solution and go to the next subproblem; otherwise branch into the subproblems again.

The essence of the branch-and-bound algorithm is to design proper relaxations for each subproblem to obtain sharper lower or upper bounds. The typical relaxation methods for INLP involve an integer relaxation of some or all of the variables and/or dropping several critical constraints [39]. Many researchers have concentrated on finding better relaxation methods, but, so far, no superior relaxation methods applicable

to the general framework have emerged [37, 40, 41]. For integer linear programming problems, integer relaxation is appropriate because the relaxed problem can be solved by well-developed linear programming methods such as the simplex method, cutting plane algorithm, or interior point method [40]. However, for the INLP case, the situation is different. If we employ integer relaxation, the relaxed subproblems become nonlinear programming problems (NLP). Solving the NLPs is also time-consuming due to the computation of derivative information on the objective and constraint functions.

If the problem under consideration is a nonconvex INLP, the situation is even worse. Each integer-relaxed subproblem of a nonconvex INLP becomes a nonconvex NLP. To guarantee the global optimality of the obtained solution, proper bounds of the nonconvex subproblems are required. For nonconvex NLPs, obtaining a global optimum which is the sharpest upper or lower bounds for the relaxed subproblems, is very difficult. So, advanced bounding techniques such as convexification, a convex under-estimator, and the cutting plane method are generally preferred in the branch-and-bound method [42, 43, 44]. However, they are inefficient and inapplicable for some nonconvex problems, because they require extensive computation to find the parameters, extra formulary manipulation of the objective and constraint functions, and excess constraints.

Recently, Tuy and Luc [44] suggested an approach for solving nonlinear programming problems with a convex objective function and monotonic nonconvex constraints. The method is based on a branch-and-bound procedure and an outer approximation on continuous space. In the algorithm, the upper bound of each subproblem is computed by polyblock approximation, where the polyblock is the union of a finite number of multidimensional hypercubes. The basic intuition of the paper is that for problems with increasing objective and constraint functions, any solution below

a feasible solution can not be a global optimum and any solution obove a infeasible solution is also infeasible. Although the problem and the method are different from ours, the theoretical derivation and the intuition of the paper inspired our proposed method.

To overcome the difficulties of nonconvex INLPs and to take advantage of the coherent property, we propose an efficient global optimization algorithm for coherent nonconvex INLPs, such as redundancy allocation problems. The method is a combination of a heuristic and a branch-and-bound method. The former is used to find the local optimum in each subproblem, and the latter is used to determine global optimality. Each subproblem is restricted by a multidimensional hypercube which is completely defined by the integer upper and lower limits of the decision variables. Sharper lower or upper bounds of the subproblems are obtained with much less computation using the coherent property. Therefore, the proposed method can find the exact solution of an INLP that has monotonic properties without any other assumptions and restrictions.

The proposed branch-and-bound approach has several advantages compared to others: 1) when solving subproblems, the proposed method uses a heuristic, instead of a relaxation, so no nonconvex NLP appears during the procedure; 2) in each iteration, the proposed algorithm finds an infeasible region using a local maximum and then eliminates this region. This search space reduction procedure is irrelevant to the convexity of the problem, so the global optimality of the solution is guaranteed; 3) by executing a heuristic on each subproblem, a local optimum is obtained. Since an incumbent solution is frequently updated, the possibility of fathoming nodes increases; 4) since the proposed methods generate multiple subproblems at the same level in the enumeration tree, it is easy to employ parallel computing.

VI.1. Theoretical Study for Global Optimization

This section explains the theoretical basis of the proposed algorithm. All of these definitions, lemmas, and theorems are used extensively throughout the rest of the following chapter.

Let $\mathcal{Z}$ be a set of integers; then $\mathcal{Z}_+ = \{x \in \mathcal{Z} | x \geq 0\}$ denotes the set of nonnegative integers and $\mathcal{Z}_{++} = \{x \in \mathcal{Z} | x > 0\}$ denotes the set of positive integers. The n tuples of $\mathcal{Z}$, $\mathcal{Z}_+$, and $\mathcal{Z}_{++}$ are denoted by $\mathcal{Z}^n$, $\mathcal{Z}_+^n$, and $\mathcal{Z}_{++}^n$, where $\mathcal{Z}_+^n$ and $\mathcal{Z}_{++}^n$ are called the *nonnegative integer orthant* and the *positive integer orthant*, respectively. For convenience, the n-dimensional i-th unit vector is denoted as $\mathbf{e}^i$.

Let $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{y} = (y_1, \ldots, y_n) \in \mathcal{Z}^n$. For any two points, $\mathbf{x}, \mathbf{y} \in \mathcal{Z}^n$, the ordering $\mathbf{x} \preceq \mathbf{y}$ implies that $\mathbf{y}$ *properly dominates* $\mathbf{x}$ and denotes $x_i \leq y_i$ for all $i = 1, \ldots, n$. The ordering $\mathbf{x} \prec \mathbf{y}$ implies $\mathbf{x} \preceq \mathbf{y}$ and $x_i < y_i$ for at least one $i = 1, \ldots, n$ and means that $\mathbf{y}$ *strictly dominates* $\mathbf{x}$. A point $\mathbf{y}$ *strongly dominates* $\mathbf{x}$ if $x_i < y_i$ for all $i = 1, \ldots, n$ and the ordering is written as $\mathbf{x} \ll \mathbf{y}$. Let $\lhd$ be any one of the orderings $\preceq$, $\prec$, or $\ll$. Then the ordering $\lhd$ satisfies the following properties:

- For all $\mathbf{z} \in \mathcal{Z}^n$, $\mathbf{x} \lhd \mathbf{z}$ and $\mathbf{z} \lhd \mathbf{y}$ implies $\mathbf{x} \lhd \mathbf{y}$;
- For all $\mathbf{z} \in \mathcal{Z}^n$, $\mathbf{x} \lhd \mathbf{y}$ implies $\mathbf{x} + \mathbf{z} \lhd \mathbf{y} + \mathbf{z}$;
- For all $\lambda > 0$, $\mathbf{x} \lhd \mathbf{y}$ implies $\lambda\mathbf{x} \lhd \lambda\mathbf{y}$.

Now, let us define *multi-dimensional segments*. For any two points $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n) \in \mathcal{Z}_+^n$ such that $\mathbf{a} \preceq \mathbf{b}$, an *n-dimensional closed segment* $[\mathbf{a}, \mathbf{b}] \subset \mathcal{Z}_+^n$ is defined as $\{\mathbf{x} \in \mathcal{Z}_+^n | \mathbf{a} \preceq \mathbf{x} \preceq \mathbf{b}\}$. Analogously, an *n-dimensional semi-closed segment* $\langle \mathbf{a}, \mathbf{b} \rangle$ and an *n-dimensional opened segment* $(\mathbf{a}, \mathbf{b})$ are defined as $\{\mathbf{x} \in \mathcal{Z}_+^n | \mathbf{a} \prec \mathbf{x} \prec \mathbf{b}\}$ and $\{\mathbf{x} \in \mathcal{Z}_+^n | \mathbf{a} \ll \mathbf{x} \ll \mathbf{b}\}$, respectively.

A function $f : \mathcal{Z}^n \to \mathcal{Z}$ is *increasing* on $\mathcal{Z}_+^n$ if for any two points $\mathbf{x}, \mathbf{y} \in \mathcal{Z}_+^n$,

$\mathbf{x} \preceq \mathbf{y}$ implies $f(\mathbf{x}) \leq f(\mathbf{y})$. Analogously, a function f is *strictly increasing* whenever $\mathbf{x} \prec (\ll)\mathbf{y}$ implies $f(\mathbf{x}) < f(\mathbf{y})$. If for any two points $\mathbf{a}, \mathbf{b} \in \mathcal{Z}_+^n$, $\mathbf{a} \lhd \mathbf{x} \lhd \mathbf{y} \lhd \mathbf{b}$ implies $f(\mathbf{x}) \leq f(\mathbf{y})$, then the function f is increasing(or strictly increasing) on an n-dimensional segment $[\mathbf{a}, \mathbf{b}] \subset \mathcal{Z}_+^n$. For increasing functions f_1 and f_2 and any nonnegative numbers λ_1 and λ_2, the function $\lambda_1 f_1 + \lambda_2 f_2$ is increasing [44].

Consider the maximization problem **INLP** where the objective and the constraint functions are increasing. Let $\mathbf{l} = (l_1, \ldots, l_n)$ and $\mathbf{u} = (u_1, \ldots, u_n)$. Let $\Omega = [\mathbf{l}, \mathbf{u}]$ and $S = \{\mathbf{x} \in \Omega | g_i(\mathbf{x}) \leq b_i, \forall i\}$, then S is the set of feasible solutions. Define the *ε-neighborhood* at $\bar{\mathbf{x}}$ as $N_\varepsilon(\bar{\mathbf{x}}) = \{\mathbf{x} \in \Omega | \mathbf{x} = \bar{\mathbf{x}} \pm \varepsilon \mathbf{e}^i, \forall i\}$. Then, a point $\bar{\mathbf{x}} \in S$ is a *global maximum* if $f(\bar{\mathbf{x}}) \geq f(\mathbf{x})$ for all $\mathbf{x} \in S$. If $\bar{\mathbf{x}} \in S$ and $f(\bar{\mathbf{x}}) \geq f(\mathbf{x})$ for all $\mathbf{x} \in S \cap N_1(\bar{\mathbf{x}})$, then the point $\bar{\mathbf{x}}$ is called a *1-neighborhood local maximum.*

Lemma 1. *If f is an increasing function on $[\mathbf{a}, \mathbf{b}] \subset \mathcal{Z}_+^n$, then $f(\mathbf{a}) \leq f(\mathbf{x}) \leq f(\mathbf{b})$ for all $\mathbf{x} \in [\mathbf{a}, \mathbf{b}]$.*

Proof. Since $\mathbf{x} \in [\mathbf{a}, \mathbf{b}]$, $\mathbf{a} \preceq \mathbf{x} \preceq \mathbf{b}$ is true. Thus, $f(\mathbf{a}) \leq f(\mathbf{x}) \leq f(\mathbf{b})$ is true by definition of the increasing function. □

Lemma 2. *In the* **INLP***, if a point $\mathbf{x}$ is in infeasible region, any point $\mathbf{y}$ which satisfies $\mathbf{y} \succ \mathbf{x}$ is in infeasible region.*

Proof. Since $\mathbf{x} \notin S$, $g_i(\mathbf{x}) > b_i$ for some i. Furthermore, $g_i(\mathbf{y}) \geq g_i(\mathbf{x})$ is true for all i because $g_i(\cdot)$ is an increasing function and $\mathbf{y} \succ \mathbf{x}$. Therefore, $g_i(\mathbf{y}) \geq g_i(\mathbf{x}) > b_i$ for some i and this implies that $\mathbf{y} \notin S$. □

Theorem 1. *In the* **INLP***, the global maximum is in a set of 1-neighborhood local maximum $T = \{\bar{\mathbf{x}} \in S | f(\bar{\mathbf{x}}) \geq f(\mathbf{x}), \mathbf{x} \in S \cap N_1(\bar{\mathbf{x}})\}$.*

Proof. Assume that $\mathbf{x}' \notin T$ is a global maximum, i.e., $f(\mathbf{x}') \geq f(\mathbf{x})$ for all $\mathbf{x} \in S$. Then, $f(\mathbf{x}') \geq f(\mathbf{x})$ for all $\mathbf{x} \in S \cap N_1(\mathbf{x}')$ is also true and it implies $\mathbf{x}' \in T$. This is a contradiction to the assumption of $\mathbf{x}' \notin T$. □

Theorem 2. *Consider the* **INLP**. *Let a point* $\bar{\mathbf{x}}$ *be a 1-neighborhood local maximum. Then,*

i) any point $\mathbf{y} \succ \bar{\mathbf{x}}$*, i.e.,* $\mathbf{y} \in \langle \bar{\mathbf{x}}, \mathbf{u}]$ *implies* $\mathbf{y} \notin S$.

ii) a global maximum exists on $\Omega \setminus \langle \bar{\mathbf{x}}, \mathbf{u}] = (\Omega \setminus [\bar{\mathbf{x}}, \mathbf{u}]) \cup \{\bar{\mathbf{x}}\}$.

Proof. i) Consider a point $\tilde{\mathbf{y}} \in N_1(\bar{\mathbf{x}})$ which satisfies $\tilde{\mathbf{y}} \succ \bar{\mathbf{x}}$. Assume that a point $\tilde{\mathbf{y}}$ is in feasible region S. Then, $f(\tilde{\mathbf{y}}) > f(\bar{\mathbf{x}})$ is true by the definition of increasing function, and this is contradiction to the assumption that $\bar{\mathbf{x}}$ is a 1-neighborhood local maximum. Therefore, $\tilde{\mathbf{y}} \notin S$ is true and any point $\mathbf{y} \succ \tilde{\mathbf{y}} \succ \bar{\mathbf{x}}$ is also in infeasible region by Lemma 2.

ii) The proof is obvious from i).

□

Theorem 3. *Consider the* **INLP**. *Let us define* $\mathbf{l}^j_{\mathbf{x}} = (x_1, \ldots, x_{j-1}, l_j, \ldots, l_n)$ *and* $\mathbf{u}^j_- = (u_1, \ldots, u_{j-1}, x_j - 1, u_{j+1}, \ldots, u_n)$. *If* $\bar{\mathbf{x}}$ *is a 1-neighborhood local maximum but not a global maximum, then the global maximum exists on* $\bigcup_{j=1}^{n} [\mathbf{l}^j_{\bar{\mathbf{x}}}, \mathbf{u}^j_-]$.

Proof. From Theorem 2 ii) a global maximum exists on $(\Omega \setminus [\bar{\mathbf{x}}, \mathbf{u}])$ if $\bar{\mathbf{x}}$ is not a global maximum. Note that

$$\begin{aligned}
\Omega \setminus [\bar{\mathbf{x}}, \mathbf{u}] &= [\mathbf{l}^1_{\bar{\mathbf{x}}}, \mathbf{u}^1_-] \cup ([\mathbf{l}^2_{\bar{\mathbf{x}}}, \mathbf{u}] \setminus [\bar{\mathbf{x}}, \mathbf{u}]), \\
&= [\mathbf{l}^1_{\bar{\mathbf{x}}}, \mathbf{u}^1_-] \cup [\mathbf{l}^2_{\bar{\mathbf{x}}}, \mathbf{u}^2_-] \cup ([\mathbf{l}^3_{\bar{\mathbf{x}}}, \mathbf{u}] \setminus [\bar{\mathbf{x}}, \mathbf{u}]), \\
&= \left(\bigcup_{j=1}^{n-1} [\mathbf{l}^j_{\bar{\mathbf{x}}}, \mathbf{u}^j_-] \right) \cup ([\mathbf{l}^n_{\bar{\mathbf{x}}}, \mathbf{u}] \setminus [\bar{\mathbf{x}}, \mathbf{u}]), \\
&= \bigcup_{j=1}^{n} [\mathbf{l}^j_{\bar{\mathbf{x}}}, \mathbf{u}^j_-],
\end{aligned}$$

as required. □

VI.2. Multi-path Branch-and-Bound Method

Initialization

At the beginning of the algorithm, the current optimal point $\mathbf{x}^*$ and value f^* are assigned as the lower limit $\mathbf{l}_{initial}$ of **INLP** and the functional value at the point, respectively. The initial feasible solution $\mathbf{x}$ is also assigned as $\mathbf{l}_{initial}$ since the proposed algorithm has no relevance to the initial point. The upper limit of **INLP** is computed by a procedure that we explain later. The detailed description of initialization is as follows:

procedure *initialize*

$\mathbf{x}^* \Leftarrow \mathbf{l}_{initial}$; $f^* \Leftarrow f(\mathbf{l}_{initial})$;

$\mathbf{u} \Leftarrow$ *compute-upperlimit*$(\mathbf{l}_{initial}, \mathbf{u}_{initial})$;

$\mathbf{l} \Leftarrow \mathbf{l}_{initial}$;

Finding the 1-neighborhood local maximum

There are many ways to find the 1-neighborhood local maximum. Finding a good local optimum in a short time is very important because the performance depends on this step. We use a steepest ascent heuristic method which is fast and produces a reasonably good solution. If the solution is a 1-neighborhood local maximum, other heuristics such as Gopal, et al. (GAG) [45], Kim and Yum (KY) [46], and others, can be used for this purpose.

The first step is finding a variable which has a maximum objective function value, where the increased variable $\mathbf{x}_{up} = \mathbf{x} + \mathbf{e}^j$ should be in feasible region S and less than the assigned upper limit $\mathbf{u}$. After finding the variable with maximum $f(\mathbf{x}_{up})$ for $j = 1, \ldots, n$, a redundancy is added to it. This procedure is repeated until the

feasibility is satisfied. The detailed algorithm is described as follows:

procedure *find-local-optimum*$(\mathbf{x}, \mathbf{u})$

while ($\mathbf{x}$ is feasible) **do**

for $j = 1$ **to** n **do**

$\mathbf{x}_{up} \Leftarrow \mathbf{x} + \mathbf{e}^j$;

if ($x_j \geq u_j$ **or** $\mathbf{x}_{up}$ is infeasible) **then** $\Delta_f(j) \Leftarrow 0$;

else $\Delta_f(j) \Leftarrow f(\mathbf{x}_{up})$;

$k \Leftarrow \arg\max_j\{\Delta_f(j)\}$; $x_k \Leftarrow x_k + 1$;

return$(\mathbf{x})$;

Upper limit computation

The most important issue in the branch-and-bound method is how to sharpen the upper bound of each subproblem. Many researchers have developed various bounding techniques for a separable INLP [34] to make the problem easier to solve. However, if the system is nonconvex and nonseparable, all known techniques for computing lower and upper limits are of no use. We compute an upper limit using the following general equation which uses constraint functions only [34].

$$x'_j = \max\{v | v \leq u_j, g_i(l_1, \ldots, l_{j-1}, v, l_{j+1}, \ldots, l_n \leq b_i, i = 1, \ldots, m, v \in Z_+\}$$

procedure *compute-upperlimit*$(\mathbf{x}, \mathbf{u})$

for $j = 1$ **to** n **do**

while ($\mathbf{x}$ is feasible **and** $x_j \leq u_j$) **do**

$x_j \Leftarrow x_j + 1$;

return($\mathbf{x}$);

Main procedure

The heart of our algorithm is in the *branch* procedure below. At the beginning of the *branch* procedure, a 1-neighborhood local maximum is found using the heuristics described in the previous section. Then the current maximum and the value at that point are updated according to the value of the 1-neighborhood local maximum. The branching is performed for each disjoint n-dimensional segment according to Theorem 3.

The whole solution space of the considered problem is bounded. This implies that the set of points in the solution space is finite because each point is a vector of integers. At each level of the enumeration tree, the solution space decreases and the global maximum exists on the union of disjoint n-dimensional segments according to Theorem 3. Since the solution space is finite, by the end of the algorithm, we find the global maximum. Note that the optimality condition does not depend on the 1-neighborhood local maximum or the corresponding methods.

Adopting the branch-and-bound method may involve selecting either branching variables or branching subproblems. Since our approach does not use specific decision variables for branching, the former situation does not apply here. For selecting branching subproblems, we use a depth first and backtracking strategy. There are three fathom rules for the new lower limit $\mathbf{l}_{new}$, the new upper limit $\mathbf{u}_{new}$, and the computed upper limit $\mathbf{u}_{computed}$: i) if the functional value of $\mathbf{u}_{new}$ is less than the current optimal value; ii) if $\mathbf{l}_{new}$ is not strictly dominated by $\mathbf{u}_{computed}$; or iii) if the functional value of $\mathbf{u}_{computed}$ is less than the current optimal value, then the subproblem is pruned permanently; otherwise the subproblem is branched into n new subproblems.

The main algorithm is fairly simple. Initialization is performed first. If the value of the computed upper limit is less than the current optimal value or the initial lower limit is infeasible, then the algorithm stops; otherwise the procedure *branch* is executed.

procedure *branch*$(\mathbf{l}, \mathbf{u})$

$\bar{\mathbf{x}} \Leftarrow$*find-local-optimum*$(\mathbf{l}, \mathbf{u})$;

if $(f(\mathbf{x}) > f^*)$ **then** update $\mathbf{x}^*$ and f^*;

for $j = 1$ **to** n **do**

$\mathbf{l}_{new} \Leftarrow \mathbf{l}^j_{\bar{\mathbf{x}}}; \mathbf{u}_{new} \Leftarrow \mathbf{u}^j_-$;

if ($\mathbf{l}_{new} \not\prec \mathbf{u}_{new}$ **or** $\mathbf{l}_{new}$ is infeasible **or** $f(\mathbf{u}_{new}) < f^*$) **then**

prune this subproblem;

$\mathbf{u}_{computed} \Leftarrow$*compute-upperlimit*$(\mathbf{l}_{new}, \mathbf{u}_{new})$;

if ($\mathbf{l}_{new} \not\prec \mathbf{u}_{computed}$ **or** $f(\mathbf{u}_{computed}) < f^*$) **then**

prune this subproblem;

else *branch*$(\mathbf{l}_{new}, \mathbf{u}_{computed})$;

procedure *branch-and-bound*

initialize;

if ($f(\mathbf{u}) < f^*$ **or** $\mathbf{l}$ is infeasible) **then stop**;

else *branch*$(\mathbf{l}, \mathbf{u})$;

VI.3. Numerical Examples

I. A bridge system $(n = 5, m = 3)$

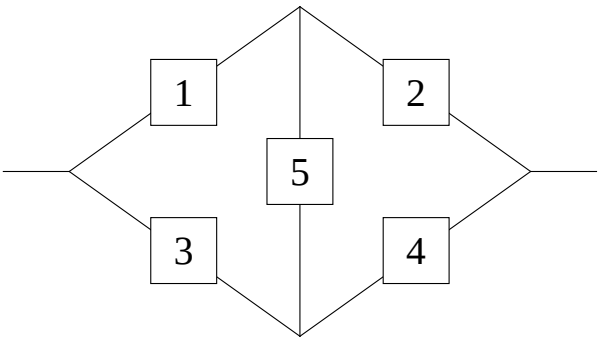

Figure 8: A Bridge System

The bridge system shown in Figure 8 is considered in order to demonstrate the proposed branch-and-bound procedures. The system consists of 5 subsystems and 3 nonlinear and nonseparable constraints. The subsystems have either parallel, options, or 2-out-of-n:G structure. The overall system reliability R_s is acquired by the pivotal decomposition method [31], where $R_j = R_j(x_j)$ and $Q_j = 1 - R_j$ for all $j = 1, \ldots, 5$. Since the overall system has a complex structure, the objective function is also nonlinear and nonseparable. A detailed definition of the problem appears below:

[**E1**]

maximize

$$R_s = R_5(1 - Q_1Q_3)(1 - Q_2Q_4) + Q_5[1 - (1 - R_1R_2)(1 - R_3R_4)]$$

subject to

$$10\exp\left(\frac{x_1}{2}\right)x_2 + 20x_3 + 3x_4^2 + 8x_5 \leq 200,$$
$$10\exp\left(\frac{x_1}{2}\right) + 4\exp(x_2) + 2x_3^3 + 6\left[x_4^2 + \exp\left(\frac{x_4}{4}\right)\right] + 7\exp\left(\frac{x_5}{4}\right) \leq 310,$$
$$12\left[x_2^2 + \exp(x_2)\right] + 5x_3\exp\left(\frac{x_3}{4}\right) + 3x_1x_4^2 + 2x_5^3 \leq 520,$$
$$(1,1,1,1,1) \preceq (x_1,x_2,x_3,x_4) \preceq (6,3,5,6,6), \mathbf{x} \in \mathcal{Z}_+^n,$$

where

$$\begin{aligned} R_1(x_1) &= (0.8, 0.85, 0.9, 0.925, 0.95, 0.975); \\ R_2(x_2) &= 1-(1-0.75)^{x_2}; \\ R_3(x_3) &= \sum_{k=2}^{x_3+1}\binom{x_3+1}{k}(0.88)^k(0.12)^{x_3+1-k}; \\ R_4(x_4) &= 1-(1-0.7)^{x_4}; \\ R_5(x_5) &= 1-(1-0.85)^{x_5}. \end{aligned}$$

The following steps depict the procedures step by step. Step 1 gives the assumptions and Step 2 is an initialization. In Steps 3 and 4, because the conditions are satisfied, the subproblem is branched into disjoint but smaller sized subproblems. On the other hand, the subproblem in Step 5 is fathomed after computing the updated current optimal solution. Finally, in Step 8, the global maximum is found. Other steps accompanied by Step 8 are executed to justify the global optimum.

1. $\mathbf{l}_{initial} = (1,1,1,1,1); \mathbf{u}_{initial} = (6,3,5,6,6);$

2. $\mathbf{x}^* = (1,1,1,1,1); f^* = 0.8733; \mathbf{u} = (5,3,5,6,6); \mathbf{l} = (1,1,1,1,1);$

3. $f(\mathbf{u}) = 0.9999 > f^*$ and $\mathbf{l}$ is feasible $\Rightarrow$ *branch*;

4. $\mathbf{x} = (2,2,4,4,2); f(x) = 0.9993 > f^*; \Rightarrow x^* = (2,2,4,4,2); f^* = 0.9993;$
   $\mathbf{l}_{new} = (2,1,1,1,1); \mathbf{u}_{new} = (5,1,5,6,6);$

$\mathbf{l}_{new} \preceq \mathbf{u}_{new}$ and $\mathbf{l}_{new}$ is feasible and $f(\mathbf{u}_{new}) = 0.9998 > f^*$;

$\mathbf{u}_{computed} = (5, 1, 4, 6, 6)$; $\mathbf{l}_{new} \preceq \mathbf{u}_{computed}$ and $f(\mathbf{u}_{computed}) = 0.9998 > f^* \Rightarrow$ *branch*;

5. $\mathbf{x} = (3, 1, 3, 5, 2); f(x) = 0.9987; \Rightarrow \mathbf{x}^* = (3, 1, 3, 5, 2); f^* = 0.9987;$

   $\mathbf{l}_{new} = (3, 1, 1, 1, 1); \mathbf{u}_{new} = (5, 0, 4, 6, 6); \mathbf{l}_{new} \not\preceq \mathbf{u}_{new} \Rightarrow$ prune this subproblem;

6. $\mathbf{l}_{new} = (3, 1, 1, 1, 1); \mathbf{u}_{new} = (5, 1, 2, 6, 6); f(\mathbf{u}_{new}) = 0.9978 < f^* \Rightarrow$ prune this subproblem;

   $\vdots$

7. $\mathbf{l}_{new} = (1, 2, 4, 1, 1); \mathbf{u}_{new} = (1, 3, 5, 3, 6);$

   $\mathbf{l}_{new} \preceq \mathbf{u}_{new}$ and $\mathbf{l}_{new}$ is feasible and $f(\mathbf{u}_{new}) = 0.9996 > f^*$;

   $\mathbf{u}_{computed} = (1, 3, 4, 3, 5); f(\mathbf{u}_{computed}) = 0.9994 > f^* \Rightarrow$ *branch*;

8. $\mathbf{x} = (1, 3, 4, 3, 3); f(x) = 0.9994; \Rightarrow \mathbf{x}^* = (1, 3, 4, 3, 3); f^* = 0.9994;$

   $\mathbf{l}_{new} = (1, 2, 4, 1, 1); \mathbf{u}_{new} = (1, 2, 4, 3, 5); f(\mathbf{u}_{new}) = 0.9981 < f^* \Rightarrow$ prune this subproblem;

   $\vdots$

The final results are $\mathbf{x}^* = (1, 3, 4, 3, 3)$ and $f^* = 0.999373$, and the remaining resources are 19.5384, 3.6496, and 35.6079 for each constraint, while $\mathbf{x}^* = (2, 2, 4, 4, 2)$ and $f^* = 0.999327$ are obtained using the GAG heuristic [45].

II. A HSP system ($n = 10, m = 2$)

The hierarchical series-parallel (HSP) system shown in Figure 9 is also tested as a numerical example. It has a nonlinear and nonseparable structure and consists of nested parallel and series systems. The system reliability function is formulated using the pivotal decomposition method [31]. The HSP system has 10 subsystems and 2

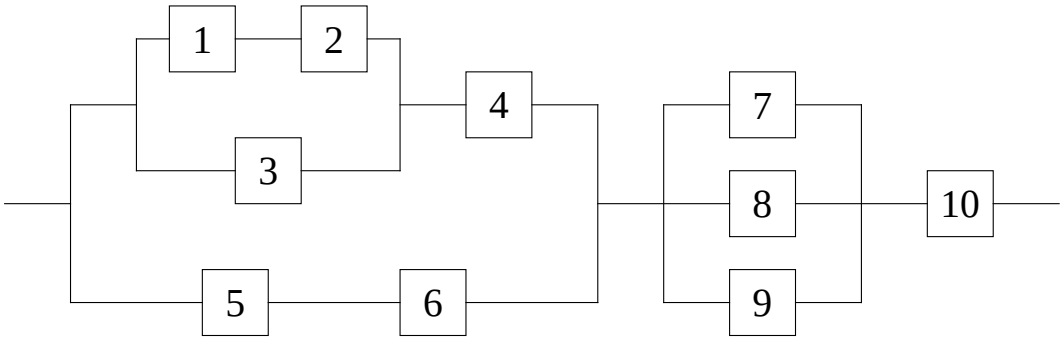

Figure 9: A HSP System

Table 2: Parameters Used in Example II

j	1	2	3	4	5	6	7	8	9	10
r_j	0.83	0.89	0.92	0.85	0.89	0.93	0.83	0.94	0.82	0.91
c_j	8	4	2	2	1	6	2	8	-	-
w_j	16	6	7	12	7	1	9	-	-	-
l_j	1	1	1	1	1	1	1	1	1	1
u_j	4	5	6	7	5	5	3	3	4	2

constrains. All related component reliabilities and the coefficients of the constraints are listed in Table 2. The maximum allowable resources b_1 and b_2 are 120 and 300, respectively. Table 2 also shows the limit of each decision variable. The mathematical descriptions and brief results are shown as follows, where $R_j = R_j(x_j) = 1-(1-r_j)^{x_j}$ and $Q_j = 1 - R_j$ for all $j = 1, \ldots, 10$.

[E2]

maximize

$$R_s = \{1 - \langle 1 - [1 - Q_3(1 - R_1R_2)]R_4\rangle(1 - R_5R_6)\}(1 - Q_7Q_8Q_9)R_{10}$$

subject to

$$\begin{aligned} c_1 \exp\left(\frac{x_1}{2}\right) x_2 + c_2 \exp\left(\frac{x_3}{2}\right) + c_3x_4 + c_4\left[x_5 + \exp\left(\frac{x_5}{4}\right)\right] & \\ +c_5x_6^2x_7 + c_6x_8 + c_7x_9^3 \exp\left(\frac{x_{10}}{2}\right) &\leq b_1, \\ w_1x_1^2x_2 + w_2 \exp\left(\frac{x_3x_4}{6}\right) + w_3x_5 \exp\left(\frac{x_6}{4}\right) + w_4x_7x_8^3 & \\ +w_5\left[x_9 + \exp\left(\frac{x_9}{2}\right)\right] + w_6x_2 \exp\left(\frac{x_{10}}{4}\right) &\leq b_2, \\ \mathbf{l} \preceq \mathbf{x} \preceq \mathbf{u} \in \mathcal{Z}_+^n. & \end{aligned}$$

After a number of steps, the global maximum $\mathbf{x}^* = (1, 1, 3, 4, 2, 1, 1, 3, 1, 4)$ and the functional value $f^* = 0.999876$ of our algorithm were found with the remaining resources of 2.4736 and 26.1036 for the two constraints. On the other hand, the GAG heuristic failed to find the global maximum ($\mathbf{x}^* = (1, 1, 1, 2, 2, 2, 1, 2, 1, 4)$, $f^* = 0.999097$).

VI.4. Numerical Experimentation

To verify the efficiency of our algorithm, we designed the following experiment. The first test system has the bridge structure shown in Figure 8 with 3 constraints. Each block in Figure 8 consists of a series system of 2, 3, 4, or 5 stages when the system has 10, 15, 20, or 25 components, respectively. The jth component reliability, r_j, is randomly generated according to an uniform distribution with a range of $[0.7, 0.9999]$. All of the coefficients of constraints, c_j, w_j, and v_j are also generated according to an uniform distribution with the range of $[1, 10]$. The mathematical description of the test problem is as follows:

[T1]

maximize

$$R_s = R_5(1 - Q_1Q_3)(1 - Q_2Q_4) + Q_5[1 - (1 - R_1R_2)(1 - R_3R_4)]$$

subject to

$$\begin{aligned} \sum_{j=1}^{n} c_j x_j &\leq U[2,3] \cdot \sum_{j=1}^{n} c_j, \\ \sum_{j=1}^{n} w_j \left(x_j + \exp\left(\frac{x_j}{4}\right)\right) &\leq (1 + e^{0.25}) \cdot U[2,3] \cdot \sum_{j=1}^{n} w_j, \\ \sum_{j=1}^{n} v_j x_j^3 &\leq 2 \cdot U[2,3] \cdot \sum_{j=1}^{n} v_j, \\ 1 \leq x_j \leq 4, x_j \in \mathcal{Z}, \qquad & j = 1, \ldots, n, \end{aligned}$$

where

$$R_j = \prod_{i=0}^{n/5-1} [1 - (1 - r_{5i+j})^{x_{5i+j}}]$$

and $Q_j = 1 - R_j$ for all $j = 1, \ldots, n$ and $U[a, b]$ is a uniform distribution on [a,b].

The second test problem is a series of a HSP system with three constraints.

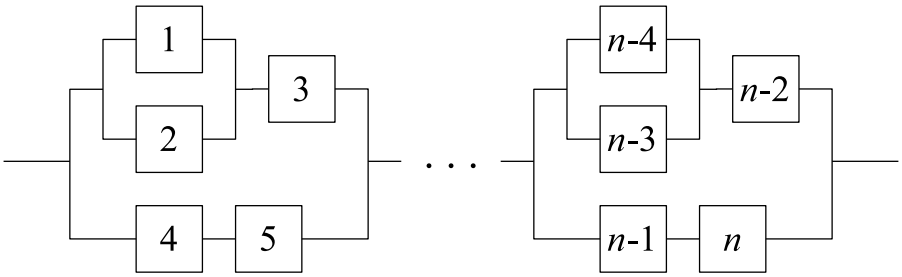

Figure 10: A Series of HSP Systems

A reliability block diagram of the HSP system is described in Figure 10 and the formulated INLP is expressed below:

[T2]

maximize

$$R_s = \prod_{k=0}^{n/5-1} \widetilde{R}_k$$

subject to

$$\begin{aligned}
\sum_{j=1}^{n} c_j x_j &\leq U[1.5, 2] \cdot \sum_{j=1}^{n} c_j, \\
\sum_{j=1}^{n} w_j x_j^2 &\leq U[2.5, 3.5] \cdot \sum_{j=1}^{n} w_j, \\
\sum_{j=1}^{n} \left[v_j x_j \exp\left(\frac{x_j}{4}\right) \right] &\leq U[1.5, 2] \cdot \sum_{j=1}^{n} v_j, \\
1 \leq x_j \leq 5, x_j \in \mathcal{Z}, &\qquad j = 1, \ldots, n,
\end{aligned}$$

where $r_i \in U[0.7, 0.9999]$ and c_i, w_i, and $v_i \in U[1, 10]$, and R_j, Q_j, and $\widetilde{R}_k$ are formulated as follows:

$$R_j = 1 - (1 - r_j)^{x_j}, \;\; Q_j = 1 - R_j,$$

$$\widetilde{R}_k = \left[1 - [1 - (Q_{k+1}Q_{k+2})R_{k+3}](1 - R_{k+4}R_{k+5})\right].$$

At the moment, the Prasad and Kuo method (PK) [35] is the best and

the Nakagawa, Nakashima, and Hattori (NNH) method [39] is the most well-known branch-and-bound method for solving the redundancy allocation problems described above. In our work, three algorithms including our proposed branch-and-bound method (BNB) were coded by C/C++, and two series of experiments for **T1** and **T2** were performed on a Pentium IV 2.4GHz PC with 512MB memory using GCC 2.95.3.

50 problems were generated for each test set of $n = 10$, 15, 20, and 25 according to the definition of **T1** and **T2**. Since all of the algorithms found the exact global optimum, the performance of each algorithm depended only on the computing time. Available bounding techniques for specific structures and assumptions were not applied in the experiment in order to demonstrate only the performance of the algorithms. Table 3 summarizes the experimental results, where each entry, (a, b, c), contains three computation times: a is the minimum, b the average, and c the maximum, respectively, on the CPU time unit (seconds on the testing platform). In the table, the first 3 columns represent 3 tested algorithms (BNB, PK, and NNH) and the last 2 columns indicate the computation time ratio of the PK and NNH to the BNB algorithm.

In the case of $n = 10$ for **T1** and **T2**, the results are not significantly different, although the BNB has the minimum average computation time. For the average computation time of **T1** at $n = 20$, the BNB is 11.07 times and 185.34 times faster than the PK and the NNH, respectively, and, at $n = 25$, the BNB is 23.41 times faster than the PK, while the NNH fails to obtain solutions for many problems in a given time. In the worst case, the BNB is 134 times (**T1**, $n = 25$) and 748.66 times (**T1**, $n = 20$) faster than the PK and the NNH, respectively. For **T2**, the results are similar to the results for **T1**, while computation times for **T2** are larger than for **T1**. From these results, we can assume that problem **T2** is harder than **T1**. Note that

Table 3: Comparison of Computation Time for Three Algorithms

T	n	BNB	PK	NNH	PK/BNB	NNH/BNB
T1	10	$(0.00, 0.01, 0.02)$	$(0.00, 0.02, 0.05)$	$(0.02, 0.03, 0.06)$	$(0.03, 1.48, 16.00)$	$(0.94, 2.97, 16.00)$
	15	$(0.03, 0.14, 0.33)$	$(0.04, 0.71, 1.72)$	$(0.91, 3.84, 9.84)$	$(0.25, 5.53, 9.99)$	$(8.35, 28.49, 61.98)$
	20	$(0.37, 2.76, 6.19)$	$(6.73, 29.66, 101.01)$	$(44.19, 487.01, 2654.05)$	$(3.57, 11.07, 23.86)$	$(18.26, 185.34, 748.66)$
	25	$(7.89, 63.22, 301.87)$	$(124.62, 1285.38, 4305.97)$	-	$(7.19, 23.41, 134.00)$	-
T2	10	$(0.00, 0.02, 0.05)$	$(0.02, 0.03, 0.06)$	$(0.03, 0.07, 0.14)$	$(0.66, 5.18, 32.00)$	$(0.98, 12.46, 94.00)$
	15	$(0.06, 0.57, 3.52)$	$(0.30, 1.42, 4.83)$	$(2.78, 10.31, 41.84)$	$(0.85, 3.44, 12.38)$	$(6.50, 26.03, 115.53)$
	20	$(0.63, 13.46, 57.92)$	$(5.80, 52.30, 117.46)$	$(241.10, 1679.63, 7979.26)$	$(1.17, 6.65, 23.34)$	$(15.54, 242.83, 1314.53)$
	25	$(25.45, 671.79, 6304.10)$	$(200.78, 4248.64, 40261.20)$	-	$(1.18, 9.20, 45.62)$	-

the ratio of the computation time between algorithms tends to increase rapidly as the size of the problem increases. In summary, the BNB is the best, the PK the second, and the NNH is the worst algorithm among these three in terms of computation time for solving redundancy allocation problems.

CHAPTER VII

HEURISTIC FOR REDUNDANCY ALLOCATION

Although recently developed exact algorithms plus the increased computing power available today have been helpful in finding exact solutions for problems in relatively shorter times, these methods are still time consuming and inapplicable to large-scale problems. As substitutes for exact algorithms, various approximation methods and heuristics have been considered to obtain near or local optimal solutions, respectively. These methods do not guarantee the global optimality of the solutions, but they can be run in a quite short time. For abundant references and methods refer to Kuo and Prasad [36] and Kuo et al. [34].

Let us focus on computational time for the moment. A fast algorithm is always better than a slow one for solving a problem, but it is not a critical factor if the application considered does not require real-time computation. In general, since the redundancy allocation problem is solved at the design stage, computation time is not a big concern, but solution quality is very important. Since contemporary systems are larger and more complicated than ever, the demand for good approximation methods and heuristics with reasonable computation times is also increasing.

Heuristic algorithms for the redundancy allocation problem can be classified into two main approaches, iterative heuristics and metaheuristics. Iterative algorithms solve problems on the basis of mathematical intuition. The solution is gradually improved at every iteration according to given factors until finally a local optimal solution is obtained. Since there is no probabilistic nature to the algorithms, the final solution of each algorithm is identical if the initial allocation and design parameters are fixed. On the other hand, metaheuristics, such as simulated annealing, genetic

algorithms, and tabu search, are based on probabilistic and artificial reasoning, and they do not require detailed mathematical information about the problem. The metaheuristics are relatively time consuming, but they obtain generally better solutions than the iterative heuristics. However, as the size of a problem increases, the efficiency of these methods decreases rapidly [46].

According to Kuo et al. [34], recent developments dealing with the redundancy allocation problem are concentrated on metaheuristics, especially genetic algorithms (GA). The main advantage of GAs is that they can manipulate very complicated problems for which solutions are hard to obtain. An outstanding GA for application to the redundancy allocation problem was developed by Coit and Smith [47, 48, 49]. They considered a series-parallel structure to maximize system reliability [47] and to minimize total cost [48]. Since GA operators, i.e., crossover and mutation, frequently generate infeasible solutions, the use of conventional GA operators is known to be inefficient in this case. To overcome this weakness, they introduced a robust adaptive penalty method based on a near-feasibility threshold (NFT) to avoid the infeasible solutions [49]. The experimental results demonstrate that a GA that employs an adaptive penalty method is superior to the surrogate constraint method [50] in terms of solution quality.

Many iterative heuristics have been developed to solve redundancy allocation problems for general and specific structures. Table 4 lists, in order of chronological development, the existing iterative heuristics for general systems. All of these heuristics gradually improve system reliability from a given initial point to a local optimum according to computed sensitivity factors that indicate the impact of each component at the current allocation. In the 1970s, research (GAG [45, 51] and NN [52, 53]) focused mainly on the development of sensitivity factors that could do a good job

Table 4: History of Iteration Heuristics for Redundancy Allocation

year	initial	developers	comment
1976	GAG	Aggarwal, et al. [45, 51]	1-neighborhood, sensitivity factor
1977	NN	Nakagawa, et al. [52, 53]	multiple balancing coefficients
1982	KI	Kohda and Inoue [54]	2-neighborhood search
1987	SHI	Shi [55]	sensitivity factors on minimal path set
1993	KY	Kim and Yum [46]	bounded infeasible region search
1996	JP	Jianping [56]	bound points search

of reflecting system structures, e.g., series, series-parallel, and complex systems. The methods that employ sensitivity factors were improved by KI [54] and SHI [55] in the 1980s. KI uses sensitivity factors but introduces the idea of deleting as well as adding redundancy to a component; this is the so-called 2-neighborhood algorithm. Shi applies a weighted sensitivity factor on a minimal path set while others compute a sensitivity factor over all of the components. A recent trend in iteration heuristics (KY [46] and JP[56]) is to search from a local optimum computed by previously developed methods, for better solutions near the boundary of the feasible region.

According to empirical experiments, there are two main factors which determine solution quality: the initial allocation and the designed sensitivity factors. It is well known that the initial allocation is crucial to the final solution of iteration heuristics. However, it is very difficult to find a proper initial allocation at the beginning of a procedure. The easiest way to subdue the effect of the initial allocation is to execute a heuristic on several randomly generated initial points and pick the best solution. This method, however, is inefficient because paths from the initial points to the final solution frequently overlap, and computation for the overlapped paths is redundant. Designing a sensitivity factor which is effective for all kinds of systems is also difficult

because there are many kinds of system structures and many types of constraints. In the case of the above listed heuristics, except for NN, there exists a unique solution path from an initial to a final solution. Since a single solution path decreases the exploitation property of the heuristics, KI, KY, and JP have tried to provide more likely solutions by repeatedly decreasing or increasing the redundancy on a current allocation. Those algorithms have generally better solutions than the classical ones, but the search region of the methods tends to be limited to the neighborhood of the first found local solution if the dimensions of a system are large.

There is a trade-off between solution quality and computation time. General rule is that the longer the algorithm's computation time, the higher the solution quality. An objective of our proposed algorithms is to improve solution quality without too much expense in computation time. As mentioned above, there are two critical factors related to solution quality. To consider these factors efficiently, we propose two new heuristics: scanning and tree heuristics. The scanning algorithm iteratively updates an incumbent solution by executing a heuristic over systematically designed initial points on each phase until no more improvement results. On every phase, a new solution space is defined by the previous incumbent solution. The tree heuristic follows the divide-and-conquer approach. The name of the tree heuristic comes from the shape of the solution paths from the initial point to several verified solutions. The main solution path is branched whenever gaps in the most likely sensitivity factors fall within a given criterion.

The proposed heuristics provide three main advantages: i) The performance of the heuristics surpasses that of any current available heuristics; ii) Since there are control parameters which control solution quality and computation time, they are suited to various applications and situations; iii) The heuristics adopt different

approaches from other existing heuristics; thus they can be easily combined with other methods. The solution quality of the combined heuristics is, in fact, markedly superior to other methods as we will demonstrate in later sections of this chapter.

VII.1. Definitions and Notation

Let $\mathcal{Z}$ be the set of integers, then $\mathcal{Z}_+ = \{x \in \mathcal{Z} | x \geq 0\}$ and $\mathcal{Z}_+^n = \{x_i \in \mathcal{Z} | x_i \geq 0, i = 1, \ldots, n\}$. For any two points, $\mathbf{x}, \mathbf{y} \in \mathcal{Z}_+^n$, the ordering $\mathbf{x} \preceq \mathbf{y}$ denotes $x_i \leq y_i$ for all $i = 1, \ldots, n$. For any two points $\mathbf{a}, \mathbf{b} \in \mathcal{Z}_+^n$ such that $\mathbf{a} \preceq \mathbf{b}$, an *n-dimensional integer segment* $[\mathbf{a}, \mathbf{b}] \subset \mathcal{Z}_+^n$ is defined as $\{\mathbf{x} \in \mathcal{Z}_+^n | \mathbf{a} \preceq \mathbf{x} \preceq \mathbf{b}\}$. A function $f : \mathcal{Z}^n \rightarrow \mathcal{Z}$ is *increasing* on $\mathcal{Z}_+^n$ if for any two points $\mathbf{x}, \mathbf{y} \in \mathcal{Z}_+^n$, $\mathbf{x} \preceq \mathbf{y}$ implies $f(\mathbf{x}) \leq f(\mathbf{y})$. A solution path $P = \{\mathbf{x}_0, \ldots, \mathbf{x}_k\}$ is defined as a sequence of points from an initial point $\mathbf{x}_0$ to $\mathbf{x}_k$.

Additional notation

$\mathbf{x}$	$(x_1, \ldots, x_n)$
$\mathbf{x}^*, f^*$	final allocation and its overall system reliability, respectively
$\mathbf{x}_{\{i+\}}$	$(x_1, \ldots, x_{i-1}, x_i + 1, x_{i+1}, \ldots, x_n)$
$\mathbf{x}_{\mathbf{y}\{i\}}$	$(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_n)$
$\mathbf{x}_{\mathbf{y}\{i+\}}$	$(x_1, \ldots, x_{i-1}, y_i + 1, x_{i+1}, \ldots, x_n)$
$F_i(\mathbf{x})$	a sensitivity factor of the ith component
F	a set of sensitivity factors, $\{F_1, \ldots, F_n\}$
$\mathbf{l}_0, \mathbf{u}_0$	initial lower and upper limits of $\mathbf{x}$, respectively
Δf_i	$f(\mathbf{x}_{i+}) - f(\mathbf{x})$

$\Delta g_j, \Delta g_{ij}$ $\quad g_j(\mathbf{x}_{i+}) - g_j(\mathbf{x})$ and $g_{ij}(\mathbf{x}_{i+}) - g_{ij}(\mathbf{x})$, respectively

Δh_j $\quad (g_j(\mathbf{x}_{\{i+\}}) - g_j(\mathbf{x}))/(b_j - g_j(\mathbf{x}))$

VII.2. Steepest Ascent Rate Heuristic Method

The simplest greedy heuristic for the redundancy allocation problem is the steepest ascent heuristic method, which iteratively increases the redundancy of a component that has a maximum sensitivity factor, where the sensitivity factors can be calculated as follows:

$$F_i(\mathbf{x}) = \begin{cases} \emptyset & \text{if } \mathbf{x}_{\{i+\}} \text{ is infeasible or } x_i = u_i \\ \Delta f_i = f(\mathbf{x}_{\{i+\}}) - f(\mathbf{x}) & \text{otherwise.} \end{cases}$$

This approach, however, is not appropriate if there are several constraints on the problem because the steepest rate does not reflect the effect of the constraints. A better approach for this case with multiple constraints is the steepest ascent rate heuristic. The sensitivity factor of this method is the marginal rate of difference between the functional values of the objective and the constraints. If the number of constraints is more than one, the maximum value is selected for each component as follows:

$$F_i(\mathbf{x}) = \begin{cases} \emptyset & \text{if } \mathbf{x}_{\{i+\}} \text{ is infeasible or } x_i = u_i \\ \dfrac{\Delta f_i}{\max\limits_j \Delta h_j} = \dfrac{f(\mathbf{x}_{\{i+\}}) - f(\mathbf{x})}{\max\limits_j \left\{ \dfrac{g_j(\mathbf{x}_{\{i+\}}) - g_j(\mathbf{x})}{b_j - g_j(\mathbf{x})} \right\}} & \text{otherwise.} \end{cases} \tag{7.1}$$

Using the above sensitivity factors, the procedures of the steepest ascent rate heuristic can be derived as shown below. The procedure and sensitivity factors

are very similar to those of GAG although GAG has more sophisticated sensitivity factors, $\Delta f_i / \max_j \{\Delta g_{ij}/(\max_i \Delta g_{ij})\}$. However, the sensitivity factor of GAG is not suitable if the system is nonseparable.

$\underline{(\mathbf{x}^*, f^*) = \textit{steepest-ascent-rate}(\mathbf{l}_0, \mathbf{u}_0)}$

Step 0. Let $\mathbf{x} = \mathbf{l}_0$.

Step 1. Compute the sensitivity factor $F_i(\mathbf{x})$ by Equation (7.1). If $F = \emptyset$, then $\mathbf{x}^* = \mathbf{x}$ and $f^* = f(\mathbf{x}^*)$. Stop algorithm.

Step 2. $\mathbf{x} = \mathbf{x}_{\{\arg\max_i \{F_i\}+\}}$. Go to **Step 1**.

The following proposition describes an important property for developing tree and scanning heuristics. The proposition and subsequent proof show that the steepest ascent rate heuristic improves its solution under a bounded area.

Proposition 1. *If an initial point is* $\mathbf{l}_0$ *and an upper limit is* $\mathbf{u}_0$*, such that* $\mathbf{l}_0 \preceq \mathbf{u}_0$*, a solution path* P *of the steepest ascent rate heuristic is a subset of an* n*-dimensional integer segment* $[\mathbf{l}_0, \mathbf{u}_0]$.

Proof. Let $P = \{\mathbf{l}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k\}$ be a solution path. Since $\mathbf{l}_0 \preceq \mathbf{x}_1 \preceq \cdots \preceq \mathbf{x}_k$ by Step 2 of the steepest ascent rate heuristic, and $\mathbf{x}_1, \ldots, \mathbf{x}_k \preceq \mathbf{u}_0$ by Equation (7.1), $\mathbf{l}_0 \preceq \mathbf{x}_1 \preceq \cdots \preceq \mathbf{x}_k \preceq \mathbf{u}_0$ is satisfied, i.e., $\{\mathbf{l}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k\} \subset [\mathbf{l}_0, \mathbf{u}_0]$ as required. □

VII.3. Tree Heuristic Method

The main weakness of heuristics employing sensitivity factors is that it is difficult to design a sensitivity factor which is suitable for all kinds of systems. Nakagawa and Nakashima [53], and subsequently Kuo et al. [52], tried to overcome this weakness

by providing diversity in sensitivity factors. NN searches a number of paths with different balancing coefficients on the same sensitivity factor equation, i.e., there are a number of different sensitivity factors. A number of local solutions are obtained by executing the same algorithm over those sensitivity factors; then, the best one among them is selected. The solution quality of this method outperforms other existing heuristics of the same type. However, the solution quality of NN does not work for large-scale problems because the exploration property of the solution paths is limited by the sensitivity factor framework.

When heuristics are executed using sensitivity factors, it often appears that several sensitivity factors are the same as, or close to, the maximum in the same iteration. These are good candidates for the global optimum as well as the maximum one. During the process, however, the existing algorithms that use sensitivity factors ignore the possibility of better solutions by only increasing the redundancy of the component with the maximum sensitivity factor. The basic idea of the proposed method, the tree heuristic, is that a sensitivity factor within some criterion that is less than, or equal to, the maximum one should be used for acquiring a better solution.

The tree heuristic is a divide-and-conquer algorithm which imitates the shape of living trees as does the branch-and-bound method. A main branch is repeatedly divided into several subbranches if some criterion is satisfied; otherwise, the main branch grows continuously without any subbranches. The criterion for branching is the gap between the sensitivity factor of each component and the maximum sensitivity factor at the current stage. The final solution is obtained by comparing the local solutions when every branch terminates.

Figure 11 is a good example for illustrating the procedures of the tree algorithm. The algorithm starts from an initial allocation of $\mathbf{l}_0$. Redundancies of elements

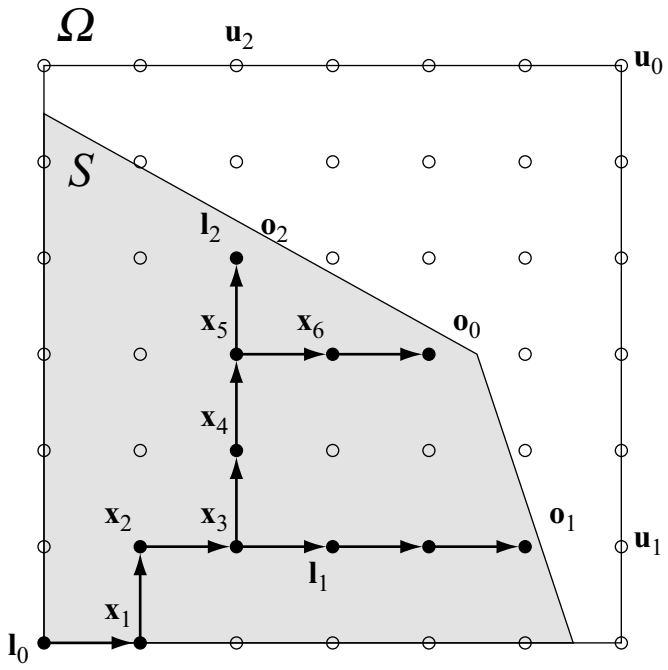

Figure 11: Graphical Example of a Tree Heuristic

are added continuously until $\mathbf{x}_3$ according to the maximum sensitivity factors. Let us assume $\mathbf{l}_1$ meets the branching criterion at point $\mathbf{x}_3$. Then, subbranch $\mathbf{l}_1$ is divided from $\mathbf{x}_0$ with solution space $[\mathbf{l}_1, \mathbf{u}_1]$. Since the solution space is not overlapped with the solution space of the main branch ($[\mathbf{x}_4, \mathbf{u}_0]$), by Proposition 1, the algorithm is efficient. The main branch is continuously increased until the second branching point $\mathbf{x}_5$. The second subbranch is $\mathbf{l}_2$ with solution space $[\mathbf{l}_2, \mathbf{u}_2]$. Local optimal solutions, $\mathbf{o}_0, \mathbf{o}_1$, and $\mathbf{o}_2$, are obtained at the terminals of each branch. Finally, $\mathbf{o}_0$ is selected as the final solution, if $f(\mathbf{o}_0) \geq f(\mathbf{o}_1) \geq f(\mathbf{o}_2)$.

The detailed procedure for the tree heuristic is described below. There are two control parameters in the procedure: α_{tree} and β_{tree}. $\alpha_{tree} \in (0, 1]$ represents the degrading rate of the maximum sensitivity factor and β_{tree} is the maximum number of subbranches at a point. If $\alpha_{tree} = 1$, the tree heuristic is identical to the steepest

ascent rate heuristic because there is no subbranch. Note that, in contrast to the branch-and-bound method, the branched solution space is not a partition of the whole solution space. Thus, this algorithm does not guarantee the global optimality of the obtained solutions.

$(\mathbf{x}^*, f^*) = \textit{tree-heuristic}(\mathbf{l}_0, \mathbf{u}_0)$

Step 0. Let $\mathbf{x} = \mathbf{x}^* = \mathbf{l}_0$ and $f^* = f(\mathbf{x}^*)$.

Step 1. Compute sensitivity factor $F_i(\mathbf{x})$ by Equation (7.1). If $F = \emptyset$, go to **Step 5**.

Step 2. $M^* = \max_i\{F_i\}$; $I^* = \arg\max_i\{F_i\}$; $F = F \setminus F_{I^*}$; $k = \beta_{tree}$.

Step 3a. If $k = 0$, go to **Step 4**, else $M = \max_i\{F_i\}$; $I = \arg\max_i\{F_i\}$; $F = F \setminus F_I$; $k = k - 1$.

Step 3b. If $M > \alpha_{tree} M^*$, then $(\mathbf{o}, f_\mathbf{o}) =$*steepest-ascent-rate*$(\mathbf{x}_{\{I+\}}, \mathbf{u}_{x_I})$;
If $f_\mathbf{o} > f^*$, then $\mathbf{x}^* = \mathbf{o}$ and $f^* = f_\mathbf{o}$. Go to **Step 3a**.

Step 4. $\mathbf{x} = \mathbf{x}_{\{I^*+\}}$. Go to **Step 1**.

Step 5. If $f(\mathbf{x}) > f^*$, then $\mathbf{x}^* = \mathbf{x}$ and $f^* = f(\mathbf{x}^*)$. Stop algorithm.

Example: A bridge system ($n = 5, m = 1, \alpha_{tree} = 0.5, \beta_{tree} = 2$)

A redundancy allocation problem [34], **E1**, with a bridge structure in Figure 8 is considered to demonstrate the procedures of the tree heuristic. The system reliability function $f(\mathbf{x})$ can be formulated by the pivotal decomposition method [31], where $R_j = 1-(1-r_j)^{x_j}$ and $Q_j = 1-R_j$ for all $j = 1, \ldots, 5$; $\mathbf{r} = (0.70, 0.85, 0.75, 0.80, 0.90)$; and $\mathbf{c}^\top = (2, 3, 2, 3, 1)$. Since the overall system has a complex structure, the objective function is also nonlinear and nonseparable.

[**E1**]

$$\max_{\mathbf{x}} f(\mathbf{x}) = R_5(1 - Q_1Q_3)(1 - Q_2Q_4) + Q_5[1 - (1 - R_1R_2)(1 - R_3R_4)]$$

$$\text{subject to} \qquad \mathbf{c}^\top \mathbf{x} \leq 20,$$

$$(1, 1, 1, 1, 1) \preceq \mathbf{x} \preceq (6, 4, 5, 6, 6), \ \ \mathbf{x} \in \mathcal{Z}_+^n,$$

Step 0. $\mathbf{x} = \mathbf{x}^* = (1, 1, 1, 1, 1)$. $f^* = 0.891325$.

Step 1. $F = \{0.2384, 0.0744, 0.2483, 0.0745, 0.0480\} \neq \emptyset$.

Step 2. $M^* = 0.2483$; $I^* = 3$; $F = \{0.2384, 0.0744, \emptyset, 0.0745, 0.0480\}$; $k = 2$.

Step 3a. $M = 0.2383$; $I = 1$; $F = F\{\emptyset, 0.0744, \emptyset, 0.0745, 0.0480\}$; $k = 1$

Step 3b. Since $0.2383 > 0.5 \times 0.2483$, $(2, 1, 1, 1, 1) \rightarrow (4, 2, 1, 1, 1) = \mathbf{o}$ and $f_{\mathbf{o}} = 0.992919$.

Since $f_{\mathbf{o}} > f^*$, then $\mathbf{x}^* = (4, 2, 1, 1, 1)$ and $f^* = 0.992919$.

Step 3a. $M = 0.0745$; $I = 4$; $F = F\{\emptyset, 0.0744, \emptyset, \emptyset, 0.0480\}$

Step 3b. Since $0.0745 < 0.5 \times 0.2483$, go to Step 4.

Step 4. $\mathbf{x} = (1, 1, 2, 1, 1)$.

Step 1. $F = \{0.0557, 0.0578, 0.0483, 0.0636, 0.0334\} \neq \emptyset$.

Step 2. $M^* = 0.0636$; $I^* = 4$; $F = \{0.0557, 0.0578, 0.0483, \emptyset, 0.0334\}$; $k = 2$.

Step 3. $k = 1$; $M = 0.0578$; $I = 2$; $(1, 2, 2, 1, 1) \rightarrow (3, 2, 2, 1, 1)$ and $f_{\mathbf{o}} = 0.993216$.

$\mathbf{x}^* = (3, 2, 2, 1, 1)$ and $f^* = 0.993216$.

Step 3. $k = 0$; $M = 0.0557$; $I = 1$; $(2, 1, 2, 1, 1) \rightarrow (3, 2, 2, 1, 1)$ and $f_{\mathbf{o}} = 0.993216$.

Step 4. $\mathbf{x} = (1, 1, 2, 2, 1)$.

Step 1. $F = \{0.0271, 0.0072, 0.0288, 0.0073, 0.0057\} \neq \emptyset$.

Step 2. $M^* = 0.0288$; $I^* = 3$; $k = 2$.

Step 3. $k = 1$; $M = 0.0271$; $I = 1$; $(2, 1, 2, 2, 1) \rightarrow (3, 1, 2, 2, 1)$ and $f_{\mathbf{o}} = 0.991361$.

Step 4. $\mathbf{x} = (1, 1, 3, 2, 1)$.

Step 1. $F = \{0.0039, \emptyset, 0.0036, \emptyset, 0.0021\} \neq \emptyset$.

Step 2. $M^* = 0.0039$; $I^* = 1$; $k = 2$.

Step 3. $k = 1$; $M = 0.0036$; $I = 3$; $(1, 1, 4, 2, 1) \rightarrow (1, 1, 4, 2, 1)$ and $f_{\mathbf{o}} = 0.99178$.

Step 3. $k = 0$; $M = 0.0021$; $I = 5$; $(1, 1, 3, 2, 2) \rightarrow (1, 1, 3, 2, 3)$ and $f_{\mathbf{o}} = 0.989329$.

Step 4. $\mathbf{x} = (2, 1, 3, 2, 1)$.

Step 1. $F = \emptyset$.

Step 5. Stop algorithm. $\mathbf{x}^* = (3, 2, 2, 1, 1)$ and $f^* = 0.993216$.

VII.4. Scanning Heuristic Method

The scanning heuristic is an iterative improvement method that uses systematically generated initial points. For all of the above heuristics listed in Table 4, the quality of the solution critically depends on the initial allocation. The first idea that was advanced to overcome the effect of the initial point was to use randomly generated initial points. However, this method is very inefficient because many solution paths

with different initial points become merged together during iteration. The main idea of the scanning heuristic, however, is a proper separation of the solution spaces for each initial point so that their solution paths rarely meet.

To explain the idea precisely, consider Figure 12. The algorithm starts at the initial point of $\mathbf{l}_0$ and a heuristic - any heuristic can be used for the purpose, but here we use the steepest ascent rate heuristic above - is executed to obtain an initial local optimum of $\mathbf{o}_0$. Since no point in the n-dimensional integer hypercube $[\mathbf{l}_0, \mathbf{o}_0]$ can have greater reliability than $f(\mathbf{o}_0)$ by the definition of the increasing function, and the solution path from $\mathbf{l}_0$ to $\mathbf{o}_0$ is in $[\mathbf{l}_0, \mathbf{o}_0]$ by Proposition 1, we can divide the solution space into several smaller solution spaces without loss of solution quality. Then, n new initial points, $\mathbf{l}_{\mathbf{o}_0\{i+\}}$ for $i = 1, \ldots, n$, are generated for the next phase, $\mathbf{l}_1$ and $\mathbf{l}_2$ in this example. Two new local optima, $\mathbf{o}_1$ and $\mathbf{o}_2$, can be obtained by executing the heuristic from the initial points $\mathbf{l}_1$ and $\mathbf{l}_2$. If we assume $f(\mathbf{o}_0) \geq f(\mathbf{o}_1) \geq f(\mathbf{o}_2)$, then the algorithm stops because no improvement has been made in this phase. On the other hand, if $f(\mathbf{o}_1) \geq f(\mathbf{o}_0) \geq f(\mathbf{o}_2)$, then the next phase is executed on $[\mathbf{l}_1, \mathbf{u}_0]$ because the reliability is updated on this phase. The algorithm stops if there is no improvement between two consecutive phases.

The following procedure and example describe the process of the scanning heuristic step by step, where Δ indicates each phase and δ_{scan} is the maximum number of phases. Since the scanning heuristic is slower than other iterative heuristics, the speed of computation can be adjusted by this control factor.

$(\mathbf{x}^*, f^*) = \textit{scanning-heuristic}(\mathbf{l}_0, \mathbf{u}_0)$

Step 0. $(\mathbf{x}^*, f^*) = \textit{steepest-ascent-rate}\ (\mathbf{l}_0, \mathbf{u}_0)$; $\Delta = \delta_{scan}$; $\mathbf{l} = \mathbf{l}_0$.

Step 1. If $\Delta = 0$, stop algorithm.

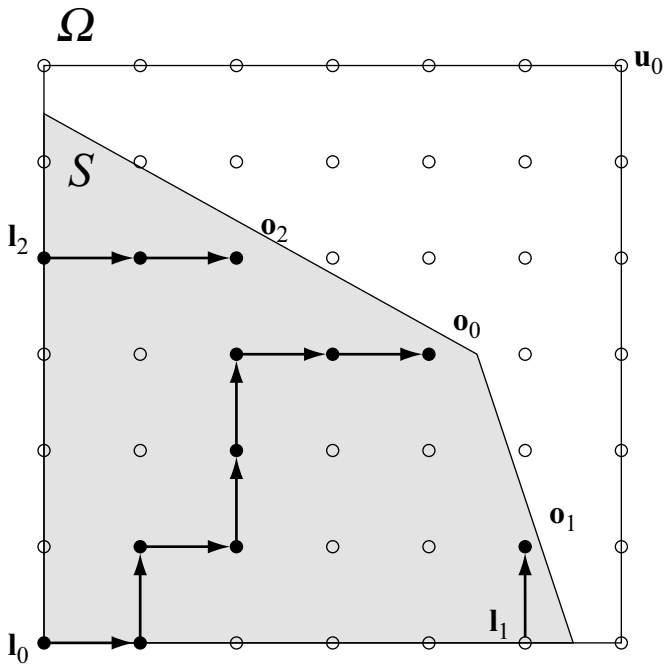

Figure 12: Graphical Example of a Scanning Heuristic

Step 2. $f_\Delta = f^*$; $\mathbf{x}_\Delta = \mathbf{x}^*$; $\Delta = \Delta - 1$; $k = 1$.

Step 3. If $k > n$, go to **Step 6**.

Step 4. $\mathbf{l} = \mathbf{l}_{\mathbf{x}_\Delta\{k+\}}$. If $\mathbf{l}$ is infeasible or $l_k > u_{0k}$, then $k = k + 1$ and go to **Step 3**.

Step 5. $(\mathbf{x}, f)$ = *steepest-ascent-rate* $(\mathbf{l}, \mathbf{u}_0)$; If $f(\mathbf{x}) > f^*$, then $\mathbf{x}^* = \mathbf{x}$; $f^* = f(\mathbf{x}^*)$; $\mathbf{l}_\Delta = \mathbf{l}$; $k = k + 1$; go to **Step 3**.

Step 6. If $f_\Delta < f^*$, then $\mathbf{l} = \mathbf{l}_\Delta$; and go to **Step 1**. Otherwise stop algorithm.

Example: A bridge system **E1** $(n = 5, m = 1, \delta_{scan} = 3)$

Step 0. $\mathbf{x}^* = (2, 1, 3, 2, 1)$; $f^* = 0.992096$; $\Delta = 3$; $\mathbf{l} = (1, 1, 1, 1, 1)$.

Step 2. $f_\Delta = 0.992096$; $\mathbf{x}_\Delta = (2, 1, 3, 2, 1)$; $\Delta = 2$; $k = 1$.

Step 3-4. $\mathbf{l} = (3, 1, 1, 1, 1)$.

Step 5. $\mathbf{x} = (3, 2, 2, 1, 1)$; $f = 0.993216$; $f^* = f$; $\mathbf{x}^* = \mathbf{x}$; $\mathbf{l}_\Delta = (3, 1, 1, 1, 1)$; $k = 2$.

Step 3-5. $\mathbf{l} = (1, 2, 1, 1, 1)$; $\mathbf{x} = (3, 2, 2, 1, 1)$; $f = 0.993216$; $k = 3$.

Step 3-5. $\mathbf{l} = (1, 1, 4, 1, 1)$; $\mathbf{x} = (1, 1, 4, 2, 1)$; $f = 0.99178$; $k = 4$.

Step 3-5. $\mathbf{l} = (1, 1, 1, 3, 1)$; $\mathbf{x} = (1, 1, 2, 3, 2)$; $f = 0.979988$; $k = 5$.

Step 3-5. $\mathbf{l} = (1, 1, 1, 1, 2)$; $\mathbf{x} = (1, 2, 3, 1, 3)$; $f = 0.990776$; $k = 6$.

Step 6. Since $f_\Delta = 0.992096 < 0.993216 = f^*$, $\mathbf{l} = (3, 1, 1, 1, 1)$.

Step 2. $f_\Delta = 0.993216$; $\mathbf{x}_\Delta = (3, 2, 2, 1, 1)$; $\Delta = 1$; $k = 1$.

Step 3-5. $\mathbf{l} = (4, 1, 1, 1, 1)$; $\mathbf{x} = (3, 2, 2, 1, 1)$; $f = 0.992919$; $k = 2$.

Step 3-4. $\mathbf{l} = (3, 3, 1, 1, 1)$ is infeasible; $k = 3$.

Step 3-5. $\mathbf{l} = (3, 1, 3, 1, 1)$; $\mathbf{x} = (3, 1, 3, 1, 2)$; $f = 0.969527$; $k = 4$.

Step 3-5. $\mathbf{l} = (3, 1, 1, 2, 1)$; $\mathbf{x} = (3, 1, 2, 2, 1)$; $f = 0.991361$; $k = 5$.

Step 3-5. $\mathbf{l} = (3, 1, 1, 1, 2)$; $\mathbf{x} = (3, 2, 1, 1, 3)$; $f = 0.998772$; $k = 6$.

Step 6. Stop algorithm. $\mathbf{x}^* = (3, 2, 2, 1, 1)$ and $f^* = 0.993216$.

VII.5. Combinations of Heuristic Methods

The heuristics mentioned above are categorized in Table 5 into several groups according to their advantages. Since the advantages of each heuristic are different, the solution quality can be improved by mixing several heuristics. As a matter of fact, various heuristics can be combined together, e.g., scanning & SHI, NN & JP, random generation & tree & KY, and so on. From many possible alternatives, we select

Table 5: Iterative Heuristic Classification by Advantage

advantage	heuristics
initial allocations	scanning, random generation
sensitivity factor design	steepest ascent rate, GAG, SHI
solution path diversity	NN, tree
fine search from a local solution	KI, KY, JP

two combinations which are easy to combine and are expected to produce a better solution quality.

Scanning and tree heuristics

The methodologies of the tree and the scanning heuristic are different: the tree heuristic increases the diversity of solution paths by using the most likely sensitivity factors and the scanning heuristic increases the solution quality by applying multiple initial points. Thus, if the two methods are combined, the main weakness of each heuristic can be effectively reduced. Since the scanning heuristic has inner routines to find local solutions from several initial points (here the *steepest-ascent-rate*) it is very easy to combine both algorithms by substituting the *tree-heuristic*, instead of the *steepest-ascent-rate*, at **Step 0** and **Step 5** in the *scanning-heuristic*.

NN and tree heuristics

The advantages of NN and the tree heuristic are similar, but they can still be combined. The tree heuristic does not depend on the sensitivity factor equation, but on a gap between sensitivity factors. Thus, the sensitivity factor equation of NN as shown

below can be used to perform the tree heuristic:

$$F_i(\mathbf{x}) = \begin{cases} \emptyset & \text{if } \mathbf{x}_{\{i+\}} \text{ is infeasible or } x_i = u_i \\ \Delta f_i \left[\alpha_{NN} \Delta x_i + (1 - \alpha_{NN}) \min_{k \in L} \Delta x_k \right] & \text{otherwise,} \end{cases} \tag{7.2}$$

where $\alpha_{NN} \in \{0, 0.1, \ldots, 0.9, 1.0, 1/0.9, 1/0.6, 1/0.3\}$ is a balancing coefficient, $\Delta x_i = \min_j\{1/\Delta h_j\}$, and $L = \{i : \Delta x_i \geq 1\}$. To apply this sensitivity factor equation, change $F_i(\mathbf{x})$ of Equation (7.1) to Equation (7.2) at **Step 1** in the *steepest-ascent-rate* and the *tree-heuristic*. The final solution can be obtained by choosing the best among the local optimal solutions of 14 different balancing coefficients α_{NN}.

VII.6. Computational Complexity of Heuristics

Let us consider the computational complexity of several iterative heuristics. To do this, some notation must be defined. Let U_i be the number of available redundancies of a component, i.e., $U_i = u_i - l_i$ for $i = 1, \ldots, n$, and U be the maximum number of U_i for $i = 1, \ldots, n$. To find the maximum or the minimum number in a set which has n elements, $O(\log n)$ operations are needed. Since each sensitivity factor of the steepest ascent rate heuristic, GAG, and NN require two minimum or maximum numbers from sets of n and m number of elements, the computational complexity of computing n sensitivity factors at a point is $O(\log n + n \log m)$. If a solution path is fixed, the computational complexity of those algorithms does not exceed the length of the solution path times $O(\log n + n \log m)$. Since the length of the solution path can not exceed nU, the computational complexity of those algorithms is $O(Un(\log n + n \log m))$. If U is bounded, then the algorithm has polynomial computational complexity.

In the case of the tree heuristic, there are at most β_{tree} branches at each node

of the main solution path. Since no subbranch divides again, at most β_{tree} times nU solution paths exist in the procedure. Thus, the computational complexity of the tree heuristic is $O(\beta_{tree}U^2n^2(\log n + n\log m))$ or $O(U^2n^2(\log n + n\log m))$ for a constant $\beta_{tree} \ll n$, because a local solution can be computed in $O(Un(\log n + n\log m))$ if we use any one of the steepest ascent rate heuristic, GAG, or NN. As for the scanning heuristic, there are n initial points at each phase. Since the algorithm executes at most δ_{scan} phases, there are at most $n\delta_{scan}$ solution paths. Thus, the computational complexity of the scanning heuristic is $O(\delta_{scan}Un^2(\log n + n\log m))$ or $O(Un^2(\log n + n\log m))$ for a constant $\delta_{scan} \ll n$. Note that only the worst case is considered when calculating the computational complexity; thus, the actual computation time may be different from the computed complexity.

Now, let us look at the computational complexity of a combination of heuristics. The combination of the scanning and tree heuristics substitutes the steepest ascent rate heuristic as the tree heuristic on the basis of the scanning heuristic. Since there are at most $n\delta_{scan}$ solution paths in the scanning heuristic and each local solution for a solution path can be computed in $O(\beta_{tree}U^2n^2(\log n + n\log m))$ using the tree heuristic, the computational complexity of the algorithm is $O(\delta_{scan}\beta_{tree}U^2n^3(\log n + n\log m))$ or $O(U^2n^3(\log n + n\log m))$ for constants, $\beta_{tree} \ll n$ and $\delta_{scan} \ll n$. The case combining the NN and the tree heuristic is more simple. Since there are 14 different balancing coefficients in the NN and the tree heuristic is applied for each coefficient, the computational complexity of the algorithm is $O(14\beta_{tree}U^2n^2(\log n + n\log m))$, i.e., $O(U^2n^2(\log n + n\log m))$ for a constant $\beta_{tree} \ll n$.

All of the above heuristics are polynomial algorithms because their complexity is bounded by a polynomial. The computational complexity of all heuristics considered depends on the number of components n, the maximum number of available redundancies U, and the number of constraints m. In general redundancy allocation

problems, since $m \ll n$ and $U \ll n$, the computational complexity of the heuristics are: $O(n^2)$ for the steepest ascent rate, GAG, and NN; $O(n^3)$ for the scanning heuristic, the tree heuristic, and the combination of the NN and the tree heuristic; $O(n^4)$ for the combination of the scanning and the tree heuristic.

VII.7. Numerical Experimentation

At the moment, the best heuristics for the general redundancy allocation problem are KY and JP using a boundary region search. The JP is eliminated from our experiments because the algorithm is very similar to KY, but KY has shown more comprehensive results when compared to other heuristics. According to the paper [46], KY outperforms SHI, KI, and simulated annealing in terms of optimality rate. Thus, SHI and KI are also excluded from our tests. To find the absolute solution qualities, the global optimum is obtained by the recently proposed lexicographical search method (LSM [35]). All algorithms are coded in C/C++ and experiments are performed on a Pentium IV 2.4GHz PC with 512MB memory after compiling by GCC 2.95.3.

The absolute solution qualities measure how close the solution of each algorithm is to the optimal solution. The average absolute error (AAE), maximum absolute error (MAE), and optimality rate (OR) are defined in percentage units as follows [46]:

$$AAE = \frac{100}{N}\sum_{j=1}^{N}\frac{R_j^* - R_j}{R_j^*},$$

$$MAE = 100\max_j\left\{\frac{R_j^* - R_j}{R_j^*}\right\},$$

$$OR = 100\frac{n\{R_j^* = R_j\}}{N},$$

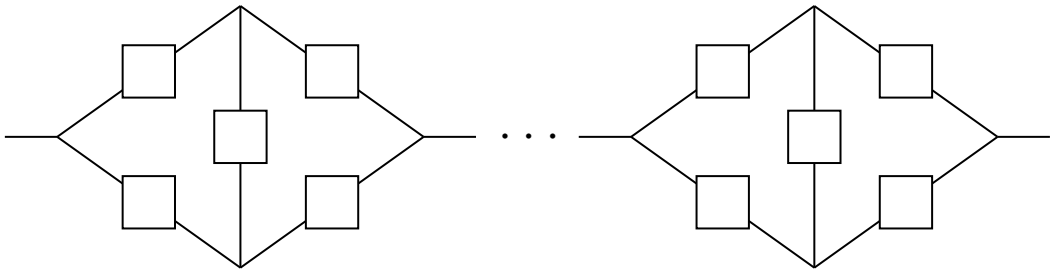

Figure 13: A Series of Bridge Systems

where where $n\{\Theta\}$ denotes the number of experiments satisfying condition Θ; N denotes the total number of experiments; R_j^* is the optimal system reliability, and R_j is the system reliability obtained by each algorithm. Since the optimal solution can not be obtained for large-scale problems, the relative solution qualities, average relative error (ARE), maximum relative error (MRE), and superior rate (SR) are also considered. ARE, MRE, and SR can be computed by replacing R_j^* in the calculation of AAE, MAE, and OR, respectively, with $\overline{R_j}$ which is the best reliability among all of the algorithms considered.

Two test problems are designed to compare the GAG, NN, KY, tree heuristic (TR), scanning heuristic (SC), a combination of the scanning and tree heuristic (SCTR), and a combination of NN and the tree heuristic (NNTR). The first test problem is a series of bridge systems depicted in Figure 13 with three nonlinear separable constraints. Each component reliability is randomly generated on the basis of an uniform distribution on interval $[0.80, 0.99]$, and the coefficients of constraint, c_j, w_j, and v_j are also generated according to an uniform distribution on intervals $[1, 10]$, $[1, 5]$, and $[1, 10]$, respectively. The mathematical description of the problem is as follows:

[T1]

$$\max_{\mathbf{x}} R_s$$

$$\begin{aligned}
\sum_{i=1}^{n} c_i x_i &\leq U[1.5, 2] \cdot \sum_{i=1}^{n} c_i, \\
\sum_{i=1}^{n} w_i \left(x_i + \exp\left(\frac{x_i}{4}\right)\right) &\leq U[1, 1.5] \cdot \sum_{i=1}^{n} w_i (1 + \exp(0.25)), \\
\sum_{i=1}^{n} v_i x_i^3 &\leq U[5, 6] \cdot \sum_{i=1}^{n} v_i, \\
1 \leq x_i \leq 5, & \quad \text{for } i = 1, \ldots, n,
\end{aligned}$$

where x_i is nonnegative integer for $i = 1, \ldots, n$; $U[a, b]$ denotes an uniform distribution on $[a, b]$; and the system reliability function is:

$$R_s = R_5(1 - Q_1 Q_3)(1 - Q_2 Q_4) + Q_5[1 - (1 - R_1 R_2)(1 - R_3 R_4)],$$

where $R_j = 1 - (1 - r_j)^{x_j}$ and $Q_j = 1 - R_j$ for $j = 1, \ldots, 5$.

The second problem tested is a series of complex systems [46] with three linear separable constraints. A block diagram of the complex system is described in Figure 14, and the formulated integer programming problem is expressed below:

[T2]

$$\max_{\mathbf{x}} R_s = \prod_{k=0}^{n/10-1} \widehat{R}_k$$

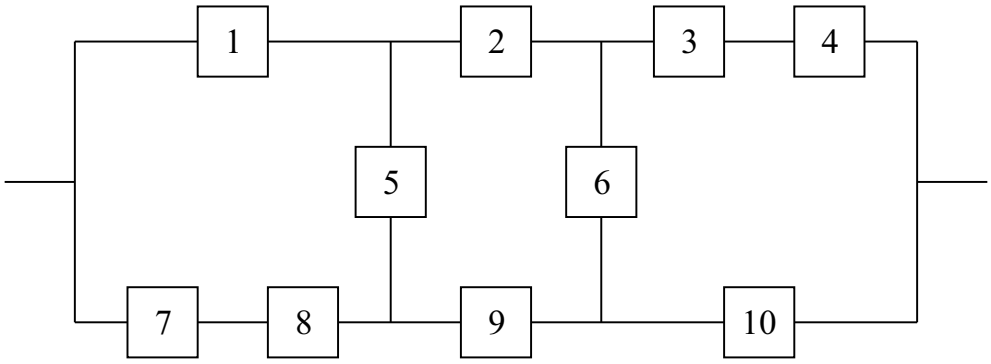

Figure 14: A Complex System

$$\begin{aligned} \text{subject to} \quad \sum_{i=1}^{n} c_i x_i &\leq U[1.5, 2] \cdot \sum_{i=1}^{n} c_i, \\ \sum_{i=1}^{n} w_i x_i &\leq U[1.5, 2] \cdot \sum_{i=1}^{n} w_i, \\ \sum_{i=1}^{n} v_i x_i &\leq U[1.5, 2] \cdot \sum_{i=1}^{n} v_i, \\ 1 \leq x_i \leq 5, &\quad \text{for } i = 1, \ldots, n, \end{aligned}$$

where x_i is a nonnegative integer; $r_i \in U[0.85, 0.95]$, $c_i \in U[1, 5]$, $w_i \in U[1, 10]$, and $v_i \in U[1, 15]$; the system reliability function can be obtained by any method in [31]:

$$\begin{aligned} \widehat{R}_k &= R_5 R_6 [1 - Q_1 (1 - R_7 R_8)](1 - Q_2 Q_9)[1 - Q_{10}(1 - R_3 R_4)] \\ &\quad + R_5 Q_6 [1 - Q_1 (1 - R_7 R_8)][1 - (1 - R_9 R_{10})(1 - R_2 R_3 R_4)] \\ &\quad + Q_5 R_6 [1 - (1 - R_1 R_2)(1 - R_7 R_8 R_9)][1 - Q_{10}(1 - R_3 R_4)] \\ &\quad + Q_5 Q_6 [1 - (1 - R_1 R_2 R_3 R_4)(1 - R_7 R_8 R_9 R_{10})] \end{aligned}$$

where $R_j = 1 - (1 - r_{10k+j})^{x_{10k+j}}$ and $Q_j = 1 - R_j$ for $j = 1, \ldots, 10$.

Two series of experiments are performed to obtain absolute performance (OR, AAE, MAE, and average computation time) and relative performance (SR, ARE, MRE, and average computation time) according to the problem types and their num-

ber of variables. For absolute performance, all testing heuristics and LSM are executed on problems with a relatively small number of elements (for **T1** $n = 5$, 10, 15, 20, and 25 and for **T2** with $n = 10$ and 20) because the optimal solution is not obtainable for problems with a large number of elements. For relative performance, all testing heuristics are applied on **T1** and **T2** with $n = 5$, 10, 15, 20, and 25. In each case, 50 randomly generated problems are tested. Tables 3 through 10 list experimental results for each measure. The solution qualities of several algorithms are controllable with control parameters. We use the following parameters:

1) NN : $\alpha_{NN} \in \{0, 0.1, \ldots, 0.9, 1.0, 1/0.9, 1/0.6, 1/0.3\}$,

2) KY : $\Delta_j = 5 \times (\Delta g_{j(1)} + \Delta g_{j(2)})$ for $j = 1, \ldots, m$,

3) SC : $\delta_{scan} = 3$,

4) TR : $\alpha_{tree} = 0.5, \beta_{tree} = 2$,

5) SCTR : $\delta_{scan} = 2, \alpha_{tree} = 0.7, \beta_{tree} = 2$,

6) NNTR : $\alpha_{NN} \in \{0, 0.1, \ldots, 0.9, 1.0, 1/0.9, 1/0.6, 1/0.3\}, \alpha_{tree} = 0.7, \beta_{tree} = 2$,

where $\Delta g_{j(1)}$ and $\Delta g_{j(2)}$ denote, respectively, the largest and the 2nd largest Δg_{ij} for $i = 1, \ldots, n$ at constraints j. Since KY improves its solution from an initial local optimum, a good initial solution is needed for the method. Although Kim and Yum [46] have used a solution of NN with $\alpha_{NN} = 0.5$, in our experiments, we select the best solution of NN with all listed α_{NN} for a better starting point. For a detailed description of KY, refer to [46].

The results of the experiments can be analyzed in three ways: number of elements, heuristic methods, and computation time. Firstly, let us examine absolute performance: OR, AAE, and MAE in Tables 6, 7, and 8. The OR critically depends on the number of elements and heuristic methods. As the number of elements increases,

OR decreases. This is reasonable because the problem is more difficult if the size of the problem is larger. The GAG has the worst solution quality overall, but it is more than 10 times faster than the second fastest method (NN). The KY always outperforms the NN in terms of absolute performance because its initial allocation is the solution of the NN. Two new proposed heuristics, the SC and TR, are superior to all existing algorithms (GAG, NN, and KY) in terms of solution quality. The heuristics that have the best solution quality are the combinations of heuristics, i.e., NNTR and SCTR. These combinations have a solution quality that is almost 90% of the OR, 0.001% of the AAE, and 0.025% of the MAE. The most important point about these heuristics is that the deviation of MAE is not large; in another words, they are robust algorithms.

The results for relative performance (of the SR, ARE, and MRE in Tables 10, 11, and 12) are similar to the results for absolute performance. Both the SR and ARE depend on the number of elements. As the problem size increases, the SR decreases significantly, the ARE decreases moderately, and the MRE shows no significant difference. On the other hand, all three measures critically depend on the heuristic methods. These results indicate that the general performance (SR and ARE) of the heuristics is related to the number of elements and methods and that the worst case performance (MRE) depends only on the heuristic methods. For all relative solution qualities, the SC and TR outperform previously developed algorithms (GAG, NN, and KY). However, contrary to previous results, the solution quality of the SCTR outperforms that of the NNTR if the problem size is large enough. When the problem size is small, the NNTR has a similar or slightly better solution quality, but the SCTR is much superior as the number of elements increases. It is also clear that the SCTR is the most robust method in terms of relative performance.

Table 6: Experimental Results of OR for Various Heuristics

T	n	GAG	NN	KY	SC	TR	SCTR	NNTR
	5	76	92	94	94	94	100	100
	10	32	70	82	84	82	90	100
T1	15	14	68	74	82	80	94	96
	20	2	34	40	56	50	78	80
	25	2	34	40	54	44	74	74
T2	10	38	54	62	88	88	98	98
	20	4	38	46	64	60	92	88
average	-	24.0	55.7	62.6	74.6	71.1	89.4	90.9

Table 7: Experimental Results of AAE for Various Heuristics

T	n	GAG	NN	KY	SC	TR	SCTR	NNTR
	5	0.0207	0.0058	0.0021	0.0019	0.0027	0	0
	10	0.0641	0.0105	0.0056	0.0031	0.0036	0.0019	0
T1	15	0.1219	0.0116	0.0072	0.0038	0.0048	0.0020	0.0010
	20	0.1587	0.0146	0.0125	0.0074	0.0067	0.0022	0.0014
	25	0.1956	0.0183	0.0163	0.0112	0.0151	0.0018	0.0033
T2	10	0.0297	0.0108	0.0077	0.0021	0.0006	0.0000	0.0007
	20	0.0656	0.0102	0.0092	0.0039	0.0038	0.0003	0.0006
average	-	0.0938	0.0117	0.0087	0.0048	0.0053	0.0012	0.0010

It is well known that there is a trade-off between the solution quality and the computation time of any algorithm. This is true for iterative heuristics used for redundancy allocation optimization. The GAG has the worst performance and the shortest computation time, while the SCTR has the best performance and the longest computation time. In fact, according to Tables 9 and 13, the solution quality critically depends on the computation time.

Table 8: Experimental Results of MAE for Various Heuristics

T	n	GAG	NN	KY	SC	TR	SCTR	NNTR
	5	0.2372	0.1854	0.0570	0.0744	0.1045	0	0
	10	0.4423	0.1997	0.1997	0.1089	0.0700	0.0700	0
T1	15	0.7232	0.1271	0.1033	0.0592	0.0716	0.0464	0.0252
	20	0.7523	0.1743	0.1743	0.0541	0.0411	0.0240	0.0291
	25	0.9670	0.1175	0.1175	0.1252	0.1382	0.0375	0.0652
T2	10	0.2717	0.1154	0.1154	0.0151	0.0177	0.0010	0.0326
	20	0.2686	0.0698	0.0698	0.0290	0.0270	0.0082	0.0177
average	-	0.5232	0.1413	0.1196	0.0717	0.0672	0.0267	0.0243

Table 9: Experimental Results of the Average Computation Time for Various Heuristics (Absolute)

T	n	GAG	NN	KY	SC	TR	SCTR	NNTR
	5	0	0.0016	0.0029	0	0	0	0.0021
	10	0.0006	0.0088	0.0144	0.0081	0.0022	0.0137	0.0318
T1	15	0.0016	0.0308	0.0473	0.0393	0.0150	0.1145	0.1569
	20	0.0066	0.0697	0.1156	0.1532	0.0522	0.7353	0.5750
	25	0.0093	0.1383	0.2187	0.3956	0.1378	2.8354	1.5395
T2	10	0	0.0032	0.0053	0.0031	0.0016	0.0069	0.0121
	20	0.0022	0.0239	0.0345	0.0540	0.0226	0.3206	0.2166

Table 10: Experimental Results of SR for Various Heuristics

T	n	GAG	NN	KY	SC	TR	SCTR	NNTR
	10	32	70	82	84	82	90	100
	20	2	42	48	60	58	88	92
T1	30	0	26	34	42	38	84	88
	40	0	12	18	36	22	76	68
	50	0	6	14	40	32	88	48
	60	0	6	8	20	14	74	50
	10	38	54	62	88	88	98	98
	20	4	40	48	66	62	94	90
T2	30	0	18	18	52	28	82	58
	40	0	6	8	40	14	82	40
	50	0	0	2	30	8	88	16
	60	0	2	6	40	8	86	18
average	-	6.3	23.5	29.0	49.8	37.8	85.8	63.8

Table 11: Experimental Results of ARE for Various Heuristics

T	n	GAG	NN	KY	SC	TR	SCTR	NNTR
	10	0.0641	0.0105	0.0056	0.0031	0.0036	0.0019	0
	20	0.1575	0.0134	0.0113	0.0062	0.0055	0.0010	0.0003
T1	30	0.2772	0.0196	0.0144	0.0073	0.0161	0.0019	0.0016
	40	0.3526	0.0327	0.0250	0.0154	0.0252	0.0038	0.0048
	50	0.4859	0.0365	0.0321	0.0161	0.0194	0.0008	0.0088
	60	0.5880	0.0369	0.0328	0.0212	0.0318	0.0031	0.0098
	10	0.0297	0.0108	0.0077	0.0021	0.0006	0.0001	0.0007
	20	0.0655	0.0101	0.0092	0.0039	0.0037	0.0002	0.0005
T2	30	0.0793	0.0178	0.0161	0.0037	0.0106	0.0011	0.0026
	40	0.1696	0.0282	0.0239	0.0090	0.0169	0.0014	0.0076
	50	0.1642	0.0408	0.0319	0.0079	0.0259	0.0006	0.0143
	60	0.2603	0.0510	0.0470	0.0090	0.0305	0.0019	0.0139
average	-	0.2245	0.0257	0.0214	0.0087	0.0158	0.0015	0.0054

Table 12: Experimental Results of MRE for Various Heuristics

T	n	GAG	NN	KY	SC	TR	SCTR	NNTR
	10	0.4423	0.1997	0.1997	0.1089	0.0700	0.0700	0
	20	0.7523	0.1743	0.1743	0.0541	0.0411	0.0240	0.0052
T1	30	0.7341	0.1253	0.1253	0.0867	0.1512	0.0312	0.0243
	40	1.4340	0.1246	0.1235	0.0798	0.1134	0.0502	0.0518
	50	2.2393	0.1723	0.1723	0.1015	0.1269	0.0164	0.0489
	60	1.5662	0.1659	0.1659	0.1104	0.1647	0.0488	0.0161
	10	0.2717	0.1154	0.1154	0.0510	0.0177	0.0010	0.0326
	20	0.2686	0.0698	0.0698	0.0290	0.0270	0.0082	0.0177
T2	30	0.4539	0.0690	0.0690	0.0383	0.0655	0.0117	0.0252
	40	0.7988	0.1514	0.1073	0.0550	0.0694	0.0176	0.0549
	50	0.5408	0.1853	0.1553	0.0394	0.1278	0.0088	0.0792
	60	1.0908	0.1593	0.1593	0.0446	0.1111	0.0265	0.0721
average	-	0.8827	0.1427	0.1364	0.0666	0.0905	0.0262	0.0432

Table 13: Experimental Results of the Average Computation Time for Various Heuristics (Relative)

T	n	GAG	NN	KY	SC	TR	SCTR	NNTR
	10	0.0012	0.0081	0.0160	0.0047	0.0025	0.0141	0.0311
	20	0.0059	0.0707	0.1121	0.1540	0.0525	0.7367	0.5759
T1	30	0.0146	0.2412	0.3796	0.9258	0.3118	8.8843	3.6104
	40	0.0363	0.5552	0.8018	3.1719	0.9529	44.245	11.402
	50	0.0690	1.1034	1.5347	8.8815	2.4690	154.75	30.765
	60	0.1172	1.8494	2.2130	18.236	4.8822	435.81	62.500
	10	0.0006	0.0031	0.0068	0.0040	0.0003	0.0072	0.0123
	20	0.0009	0.0255	0.0353	0.0553	0.0212	0.3202	0.2172
T2	30	0.0053	0.0856	0.1191	0.3802	0.1350	4.4813	1.4959
	40	0.0129	0.1893	0.2649	1.2212	0.3919	21.171	4.6044
	50	0.0228	0.3592	0.4778	3.4736	0.9582	74.291	11.699
	60	0.0390	0.5959	0.7662	6.8493	1.8187	178.03	22.564

CHAPTER VIII

CASE STUDY: REDUNDANCY ALLOCATION OPTIMIZATION FOR MEMORY INTEGRATED CIRCUITS

Consider a 1Gb DRAM [13] whose system block diagram appears in Figure 15. The memory semiconductor integrated circuit (IC) consists of two separate HALFs, and each HALF contains corresponding peripheral circuits, 8 memory blocks with a memory size of 128MB, and a number of block redundancies. Each 128MB memory block contains 32×16 memory segments with a memory size 256Kbit, and each memory segment contains 512 word lines and 512 bit lines. Each block redundancy consists of four 256Kbit arrays in a row, and each can replace any for adjacent memory segments in a row in the HALF.

VIII.1. Optimization Problems

The yield and reliability for this IC can be computed as in Table 1. However, since the architecture of the IC is different than the architecture of the IC in Chapter IV, some of the yield and reliability formula in Table 1 should be modified. The IC under consideration has two HALFs which are identical to each other. Thus, the following modifications are required in Table 1:

$$Y_{IC} = \left[Y_{PC} \cdot (Y_{MB})^{N_{mb}} \cdot Y_{BF}\right]^{N_{half}} \cdot Y_{BI}, \tag{8.1}$$

$$Y_{ICN} = \left\{ Y_{PC} \cdot \left[\prod_{i \in I} \left(1 + \frac{\lambda_i}{\alpha_i}\right)^{-\alpha_i} \right]^{N_{mb}} \right\}^{N_{half}}, \tag{8.2}$$

$$R_{IC} = (R_{NC})^{N_{half}} \cdot \sum_{n_f=0}^{N_{wlc}} \left\{ \left[\prod_{i=1}^{n_f} \frac{(N_{wlc} + 1 - i)}{N_{wlc}} \right] \Pr\{N_f = n_f\} \right\}. \tag{8.3}$$

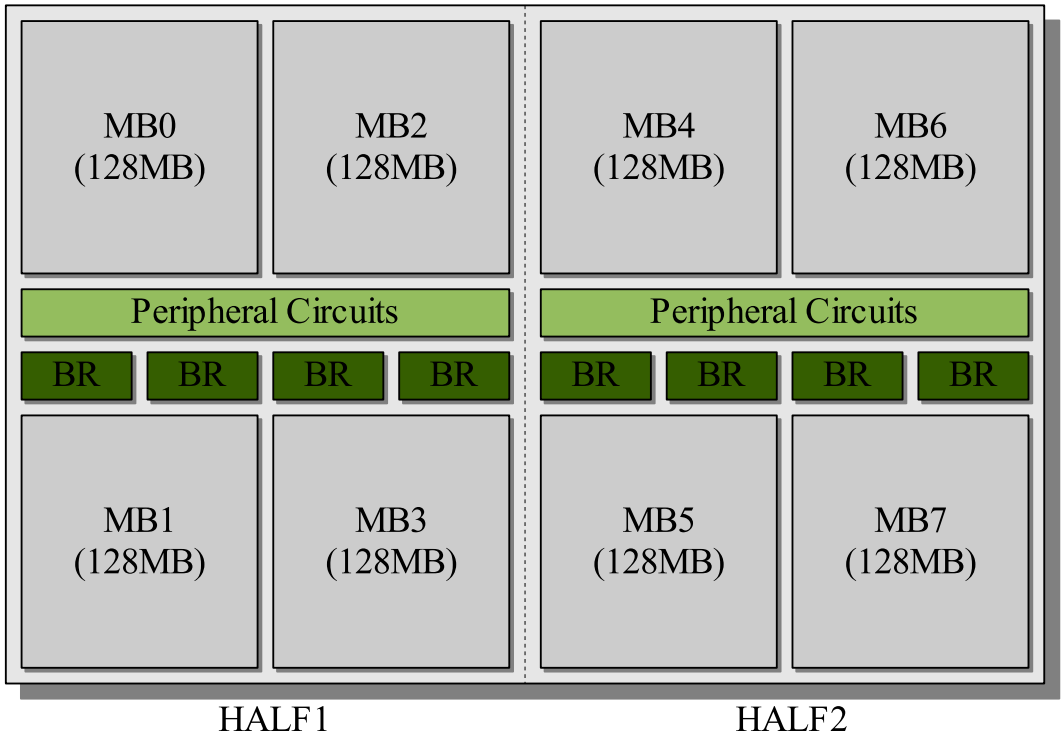

Figure 15: A Block Diagram of a 1Gb DRAM

Various types of optimization problems can be defined, for example, yield maximization, reliability maximization, manufacturing cost minimization, total profit maximization, and so on. In this chapter, we confine our concerns to a simple redundancy allocation problem which maximizes yield as follows:

[YM]

$$\begin{aligned} \text{maximize} \quad & Y_{IC}(\mathbf{n}) \\ \text{subject to} \quad A_{IC}(\mathbf{n}) &\leq \gamma_{area} A_{IC}(\mathbf{0}), \\ n_{wl}, n_{bl} &\in \{0, 2, 4, 6, \ldots 30\}, \\ n_{br}, n_{bwl}, n_{bbl} &\in \{0, 2, 4, 6, \ldots 10\}, \\ n_{ecc} &\in \{0, 1\}, \end{aligned}$$

where $\mathbf{n} = (n_{br}, n_{wl}, n_{bl}, n_{bwl}, n_{bbl}, n_{ecc})$, $\mathbf{0} = (0, 0, 0, 0, 0, 0)$, γ_{area} is the area increasing ratio, and $A_{IC}(\mathbf{n})$ is the total area of the IC for fault-tolerant allocation $\mathbf{n}$. Fault-tolerant schemes increase the $A_{IC}(\mathbf{n})$ as well as the manufacturing yield and reliability of the IC. Since $A_{IC}(\mathbf{n})$ is a function of the number of redundancies and the existence of ECC, it can be calculated as follows:

$$\begin{aligned} A_{IC}(\mathbf{n}) &= N_{half}(A_{PC} + N_{mb} A_{MB} + n_{br} A_{BR}), \\ A_{MB} &= A_{MB0} + n_{wl} A_{WL} + n_{bl} A_{BL} + n_{ecc} N_{bl} N_{par} A_{BL}, \\ A_{BR} &= A_{BR0} + n_{bwl} A_{BWL} + n_{bbl} A_{BBL} + n_{ecc} N_{bbl} N_{par} A_{BBL}, \end{aligned}$$

where A_{PC} is the area of peripheral circuits per HALF; A_{MB} and A_{MB0} are the area of a memory block with and without fault-tolerance, respectively; A_{BR} and A_{BR0} are the area of a block redundancy with and without fault-tolerance, respectively; A_{WL} and A_{BL} are the area of a word line and a bit line in a memory block, respectively; A_{BWL} and A_{BBL} are the area of a word line and a bit line in a block redundancy,

Table 14: Basic Parameters for Calculating Yield and Reliability

Parameter	Value	Comment
α_i	5	clustering factor [2]
γ	0.005	ratio constant for burn-in yield [24]
A_{IC}	$287mm^2$	chip size at introduction (2004) [2]
γ_m	72.6 %	memory area ratio (2004) [2]
μ_b	216 FIT	MIL-HDBK-217 for a 64K DRAM [14]
t_0	100000 hours	$\approx$ 11.4155 years

respectively; N_{half} is the number of HALF; n_{br} is the number of block redundancies; n_{wl} and n_{bl} are the number of row and column redundancies in a memory block; n_{bwl} and n_{bbl} are the number of row and column redundancies in a block redundancy; n_{ecc} is a binary number which represent the existence of an ECC.

VIII.2. Numerical Experimentation

To solve the redundancy allocation problem of **YM** numerically, all of the parameters in the equations should be determined. The values of some of the basic parameters are summarized in Table 14 with corresponding references. One of the important parameters is the average number of defects, λ, which can be computed by Equation (3.10). Since the defect density D_0 is a given number, we should know the critical area for obtaining the λ of each memory failure. In this experiment, we assume these as follows:

- $A^c_{PC} = A^c_{DC} = 1\ mm^2/\text{IC}$
- $A^c_{SWL} = A^c_{SBL} = 50\ mm^2/\text{IC}$

- $A^c_{DWL} = A^c_{DBL} = A^c_{CL} = A^c_{BF} = 10\ mm^2$/IC
- $A^c_{SC} = 300\ mm^2$/IC,

where A^c_Φ denotes the critical area for area Φ. A^c_{PC} and A^c_{DC} are typically designed more loosely than the other critical areas to avoid chip-kill failures on these circuit areas. Note that the sum of all of the critical areas can exceed the total area of the IC since the IC has multiple layers. Parameter γ_m is the ratio of memory area to the total area of a IC, i.e., $\gamma_m = (N_{half} N_{mb} A_{MB})/A_{IC}$. Other parameters related to the size can be calculated as follows:

- $N_{half} = 2$ HALF/IC
- $N_{mb} = 4$ MB/HALF
- $N_{bm} = 32$ rows $\times$ 16 columns/4 arrays = 128 block modules/MB
- $N_{wl} = 32$ rows $\times$ 512 word lines = 16384 word lines/MB
- $N_{bl} = 16$ columns $\times$ 512 bit lines = 8192 bit lines/MB
- $N_{tbw} = 137$ bits/memory word
- $N_{cbw} = 1$ bit/memory word
- $N_{abw} = 128$ bits/memory word
- $N_{par} = N_{tbw} - N_{abw} = 9$ bits/memory word
- $N_{mw} = N_{bl}/N_{abw} = 64$ memory words/word line
- $N_{wlc} = N_{half} \cdot N_{mb} \cdot N_{wl} \cdot N_{mw} = 8388608$ memory words/IC
- $N_{bwl} = 512$ word lines/BR

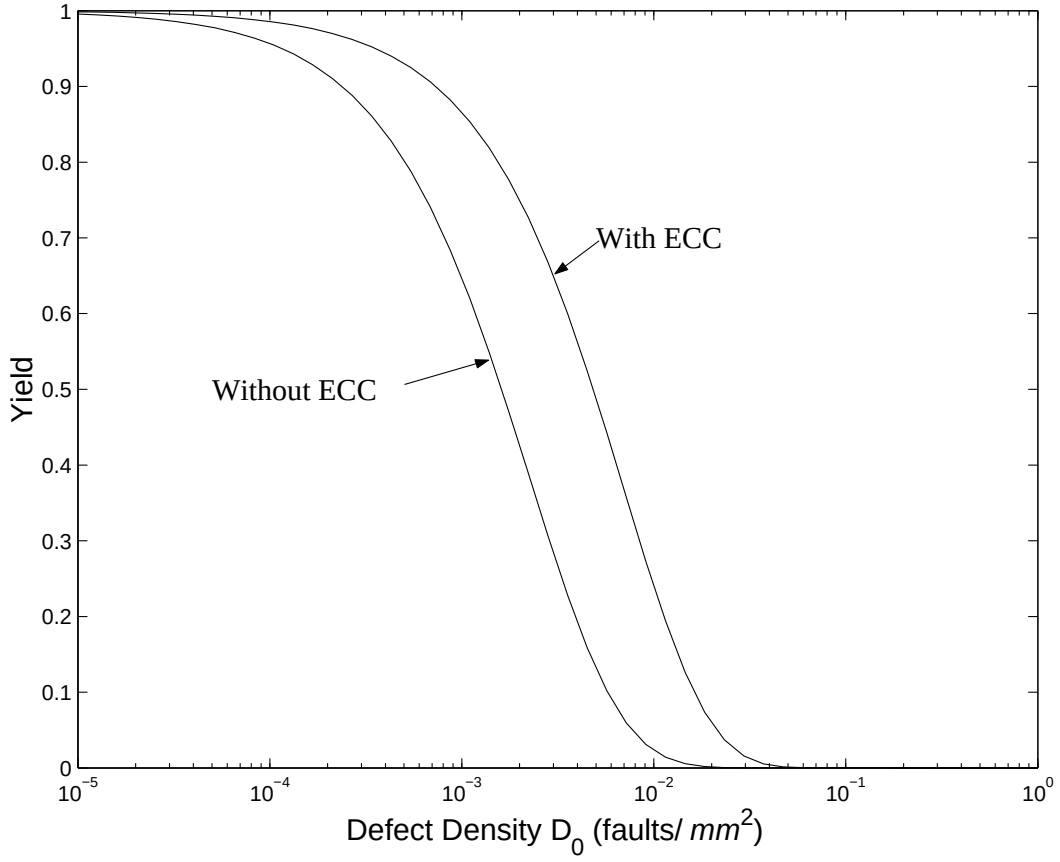

Figure 16: Comparison of IC Yield Related to the Existence of ECC

- $N_{bbl} = 4$ columns $\times$ 512 bit lines = 2048 bit lines/BR

According to the ITRS road map, the random defect density, D_0, is expected to be 2748 faults/m^2 (0.002748 faults/mm^2) by 2004. In several series of our experiments, the defect density, D_0, varies from 0.00001 faults/mm^2 (0.00287 faults/IC) to 10 faults/mm^2 (287 faults/IC) and the area increasing ratio, γ_{area}, varies from 1.01 to 1.09, i.e., an increase from 1 to 9 % of the total IC area without fault-tolerance.

On the basis of the above defined parameters, several series of experiments for the proposed redundancy allocation algorithms, BNB, TREE, and SCAN, are performed on a Pentium IV 2.4GHz PC with 512MB memory. In each series of

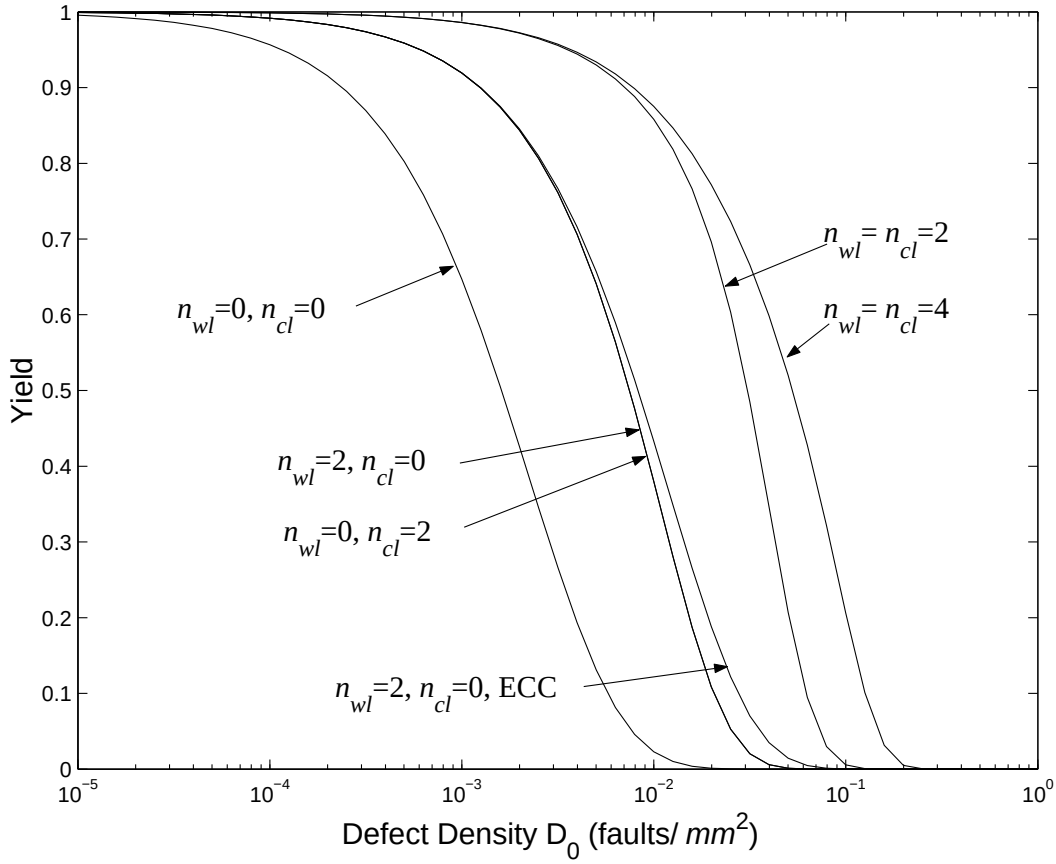

Figure 17: Comparison of IC Yield Related to the Various Row and Column Redundancy Policies

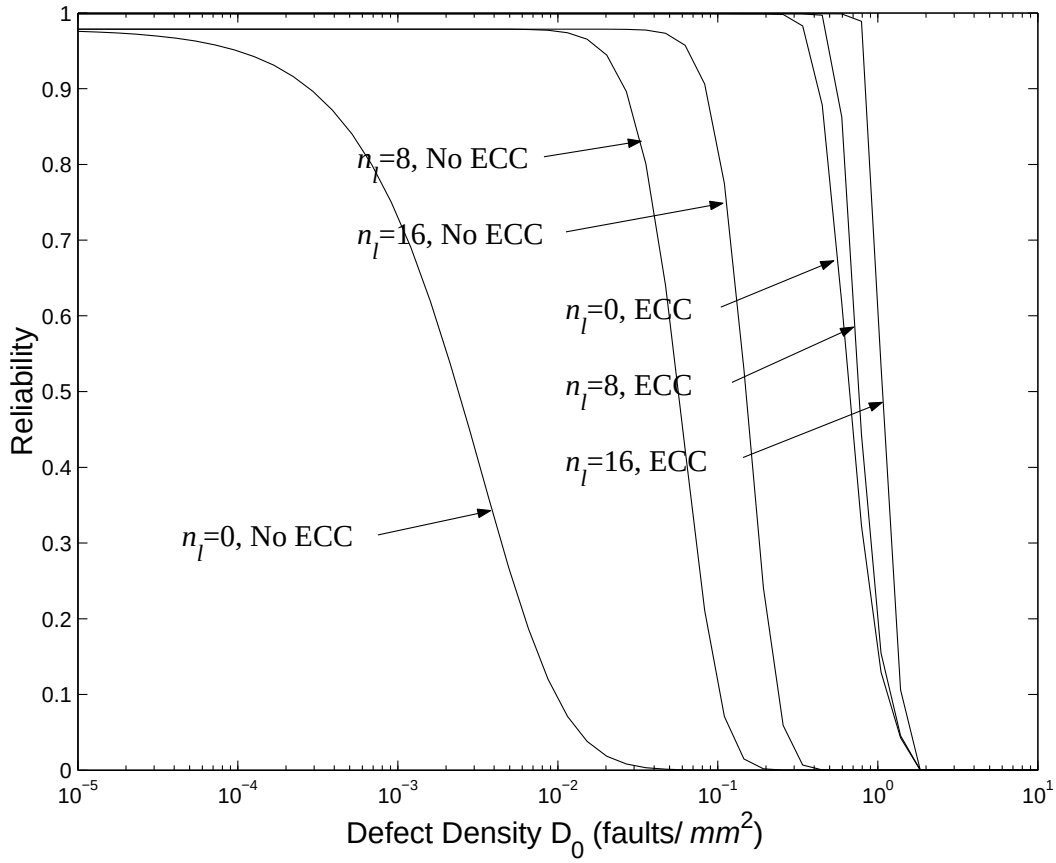

Figure 18: Comparison of IC Reliability Related to the Various Row and Column Redundancy Policies

experiments, BNB and SCAN always find global optimum, and TREE rarely failed to find optimal solutions. The reason for the good performance of all of the algorithm is the small size of **YM**, and we expect that the results would be different if the problem size were larger.

The first series of experiments is performed to investigate the effect of the ECC. Figure 16 shows yield variation related to defect density, D_0, where no redundancies are employed. At $D_0 = 0.001$ faults/mm^2, the yield of the IC with ECC is 0.8658 and the yield of the IC without ECC is about 0.6470, which is about 21% yield difference. The gap between these yields decreases as the defect density either gets close to zero or close to a large number.

The yield also varies depending on the number of redundancies. The calculated yields of the IC are shown in Figure 17 for various fault-tolerant policies. As the number of redundancies increases, the yield also increases. At $D_0 = 0.001$ faults/mm^2, the yield for $n_l = n_{wl} + n_{cl} = 0$ is 0.6470, the yield for $n_l = 2$ is 0.9190, the yield for $n_l = 4$ is 0.9859, the yield for $n_l = 8$ is 0.9860, and the yield for both $n_l = 2$ and ECC is 0.193. At $D_0 = 0.01$ faults/mm^2, the yield for $n_l = 0$ is 0.0229, the yield for $n_l = 2$ is 0.3800, the yield for $n_l = 4$ is 0.8580, the yield for $n_l = 8$ is 0.8752, and the yield for both $n_l = 2$ and ECC is 0.4330. At some point, the yield does not significantly improve even though more redundancies are added. For example, the yields for $n_l = 4$ and $n_l = 8$ are similar if the defect density is less than 0.01 faults/mm^2. Another important point to emphasize is the effect of ECC is not so significant if there exist some line redundancy. Compare the Figure 16 to Figure 16. If the number of line redundancies increase, the effect of ECC decrease even more. The reason for this phenomenon is that remaining line redundancies repair almost single cell failures if enough line redundancies are provided.

Since the defect-based reliability model in Chapter IV depends on the number of carry-over single cell failures, the reliability of the IC also relies on the defect density, D_0. Figure 18 shows that the reliability variations for the various fault-tolerant policies related to defect density. Since the redundancies cover carry-over single cell failures, enough redundancies significantly improves the reliability. More dramatic result can be observed by employing ECC. ECC improves the high reliable range as well as the reliability itself.

The second series of experiments is designed to examine the optimal number of redundancies and the resulting effects on yield and reliability. Table 15 shows optimal yield, corresponding reliability, and optimal allocation of decision variable for various defect densities and constraints with γ_{area}. Figure 19 shows optimal yields for various γ_{area} related to defect density. In Figure 19 and Tables 15, the condition 'No FT' denotes that neither a redundancy and nor ECC are employed; condition 'ECC only' indicates that ECC only is employed; and condition γ_{area} indicates optimal solutions for constraints with γ_{area}. For all of the defect densities under consideration, optimal yields are obviously better than the yields for conditions No FT, ECC only, and combination of $n_{wl} = n_{bl} = 4$ and ECC. For all conditions γ_{area}, the optimal yields are very similar but only tail shapes are different. However, if the condition γ_{area} is more tight, the results may differ. Note that ECC occupies a large amount of area (13.9179 mm^2), so ECC can not be employed if γ_{area} is less than 1.0485. This implies that the optimized number of redundancies can improve yield better than an ECC. The trend can be observed in Table 15. In that table, the optimal solutions for $\gamma_{area} = 1.05$ do not employ ECC even though there is enough space.

In summary, although both redundancy and ECC affect both yield and reliability, the number of redundancies critically affects yield and ECC critically affects the reliability of the IC. Thus, both fault-tolerant policies are required to ensure certain

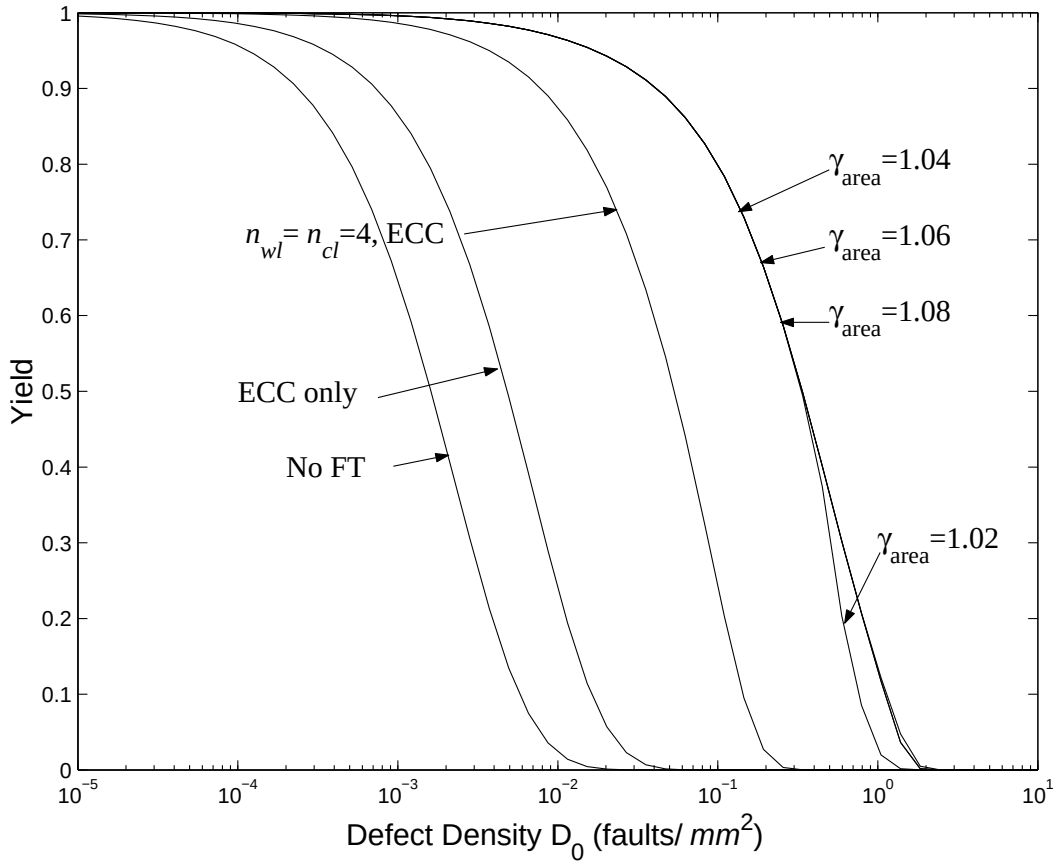

Figure 19: Comparison of Optimal Yield Related to the Various Conditions

levels of manufacturing yield and reliability in semiconductor products. In addition, it is also very important to find the optimal number of redundancies to minimize the area of a chip because yield and reliability are not enhanced significantly if enough redundancies are provided to the IC.

Table 15: Optimal Allocations for the Various Conditions

D_0	conditions	Y_{IC}	R_{IC}	$(n_{br}, n_{wl}, n_{bl}, n_{bwl}, n_{bbl}, n_{ecc})$
0.005	No FT	0.13153630	0.26358501	(0, 0, 0, 0, 0, 0)
	ECC only	0.48838501	0.99892040	(0, 0, 0, 0, 0, 1)
	$\gamma_{area} = 1.01$	0.98186544	0.97867271	(4, 12, 12, 2, 2, 0)
	$\gamma_{area} = 1.03$	0.98186544	0.97867271	(4, 12, 12, 2, 2, 0)
	$\gamma_{area} = 1.05$	0.98186544	0.97867271	(4, 12, 12, 2, 2, 0)
	$\gamma_{area} = 1.07$	0.98186544	0.97867271	(4, 12, 12, 2, 2, 0)
	$\gamma_{area} = 1.09$	0.98186544	0.97867271	(4, 12, 12, 2, 2, 0)
0.01	No FT	0.02289943	0.09333350	(0, 0, 0, 0, 0, 0)
	ECC only	0.24011778	0.99891993	(0, 0, 0, 0, 0, 1)
	$\gamma_{area} = 1.01$	0.96734620	0.97867271	(6, 14, 14, 2, 2, 0)
	$\gamma_{area} = 1.03$	0.96734620	0.97867271	(6, 14, 14, 2, 2, 0)
	$\gamma_{area} = 1.05$	0.96734620	0.97867271	(6, 14, 14, 2, 2, 0)
	$\gamma_{area} = 1.07$	0.96734620	0.99892058	(6, 14, 14, 2, 2, 1)
	$\gamma_{area} = 1.09$	0.96734620	0.99892058	(6, 14, 14, 2, 2, 1)
0.5	No FT	0.00000000	0.00000000	(0, 0, 0, 0, 0, 0)
	ECC only	0.00000000	0.00000000	(0, 0, 0, 0, 0, 1)
	$\gamma_{area} = 1.01$	0.31677388	0.97866823	(6, 16, 14, 2, 8, 0)
	$\gamma_{area} = 1.03$	0.36431155	0.97866823	(10, 30, 30, 2, 4, 0)
	$\gamma_{area} = 1.05$	0.36431155	0.97866823	(10, 30, 30, 2, 4, 0)
	$\gamma_{area} = 1.07$	0.36431322	0.99891822	(10, 30, 30, 2, 4, 1)
	$\gamma_{area} = 1.09$	0.36431322	0.99891822	(10, 30, 30, 2, 4, 1)

CHAPTER IX

CONCLUDING REMARKS

Nanotechnology is an engineering area that is currently attracting widespread attention. It is anticipated that nanotechnology will revolutionize many industries, as it is applied to various real-world applications. Nanomanufacturing involves varied disciplines including accurate modeling and optimization. In this dissertation, I have investigated semiconductor manufacturing, which represents the first realistic application of nanotechnology to date. Yield and reliability are the most important measures for manufacturability and productivity. The main goal of this study is to build integrated mathematical models for yield and reliability and to maximize the yield and reliability by developing efficient optimization methodologies.

Due to imperfections in the manufacturing process, semiconductor integrated circuits intrinsically possess many defects and faults, which lead to the failure of semiconductor products and subsequently, to yield loss and degradation of reliability. The defects or faults can be managed by eliminating the critical defects or by designing robust systems for dealing with the defects, so-called fault-tolerant design. For these approaches, accurate defect-based yield and reliability models are essential because these models will enable us to investigate the failure sources and to estimate yield and reliability without volume production. There are many yield and reliability models for semiconductor integrated circuits. However, most models at present consider only some of the factors that affect yield and reliability. I have developed integrated yield and reliability models that consider defect types, burn-in effects, and fault-tolerant schemes with both hierarchical redundancy and error correcting codes, simultaneously. These models are expected to estimate yield and reliability more accurately

than others, and they can be employed for optimizing the yield and reliability of memory semiconductor integrated circuits.

Theoretically and practically, increasing the functionality and the complexity of systems decreases yield and reliability. The most efficient way to improve the yield and reliability of systems without high additional costs is to add redundancy onto unreliable components and to optimize the number of redundancies. In general, yield and reliability optimization problems can be formulated as nonconvex integer or mixed-integer nonlinear programming problems with coherent properties. In this study, I have developed a global optimization algorithm using the branch-and-bound method and two multi-path iterative heuristics, tree and scanning heuristics, which are very efficient for redundancy allocation problems in terms of both solution quality and computation time. These methods can also be directly employed to optimize the yield and reliability of memory semiconductor integrated circuits with fault-tolerant schemes which were previously investigated.

Using the current research as above, the following research directions should be considered for future research:

Yield and reliability enhancement of semiconductor products

Yield and reliability can be enhanced further by employing other techniques. Some good candidate include, 1) conjunction of cost and profit models to the current model, 2) efficient redundant system design, 3) burn-in time and level optimization, and 4) layout and conductive line optimization. In addition, statistical process control for monitoring and diagnosing manufacturing variation is a significant defect management approach for enhancing yield and reliability in semiconductor products.

Development of new optimization algorithms

Since systems at nano and quantum scales employ Quantum Physics, new optimization problems are expected in these emerging systems. On the basis of the new physics, a new paradigm for optimization algorithms may be required to solve newly emerging problems.

Micro/nano manufacturing and fabrication

There are many candidates for nanofabrication methods, but none of them is dominant so far. Since nanofabrication facilities are very high in cost, setting up volume production without significant verification of the methods is a great risk. The verification of the productivity and manufacturability of newly developed technologies using models, simulation, and optimization is essential in advance of volume production.

Nano/quantum reliability

Current reliability theory and failure mechanisms may not be directly applicable to nano-systems and quantum-systems. On the atomic or molecular level, defects, faults, failures, wear-out, and repair mechanisms may have different physical meanings. Quantum physics and statistics may need to be employed to explain the correlations between current systems and nano/quantum-systems. This may increase the demand for new paradigms of nano/quantum reliability.

Nanotechnology and semiconductor engineering are relatively new to industrial engineering. In fact, most research on these subjects has been accomplished by other engineering fields such as chemistry, mechanics, and electronics. I believe the main role of industrial engineering is to make connections between science, engineering, economics, and management, and it is a very important job. Since these emerging engineering fields are multidisciplinary property, there are plenty of research topics in these new areas which can be most successfully investigated by industrial engineers.

REFERENCES

[1] J. R. Heath, P. J. Kuekes, G. S. Snider, and S. Williams, "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol. 280, pp. 1716–1721, 1998.

[2] ITRS, "International technology roadmap for semiconductors: 2002 update," International Technology Roadmap for Semiconductors, Tech. Rep., 2002, http://public.itrs.net.

[3] K. Chakraborty and P. Mazumder, *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories*. Upper Saddle River, NJ: Prentice Hall PTR, 2002.

[4] W. Kuo, W. K. Chien, and T. Kim, *Reliability, Yield, and Stress Burn-in: A Unified Approach for Microelectronics Systems Manufacturing and Software Developement*. Norwell, MA: Kluwer Academic Publishers, 1998.

[5] J. P. Gyvez and D. K. Pradhan, *Integrated Circuit Manufacturability: The Art of Process and Design Integration*. New York: IEEE Press, 1999.

[6] I. Koren and Z. Koren, "Defect tolerance in VLSI circuits: Techniques and yield analysis," in *Proceedings of the IEEE*, 1998, vol. 86, no. 9, pp. 1819–1837.

[7] R. Doering and Y. Nishi, "Limits of integrated-circuit manufacturing," in *Proceedings of the IEEE*, 2001, vol. 89, no. 3, pp. 375–393.

[8] R. W. Keyes, "Fundamental limits of silicon technology," in *Proceedings of the IEEE*, 2001, vol. 89, no. 3, pp. 227–239.

[9] M. Quirk and J. Serda, *Semiconductor Manufacturing Technology*. Upper Saddle River, NJ: Prentice Hall, 2001.

[10] *Military Standard: Definitions of Terms for Reliability and Maintainability*. Washington, DC: Department of Defense, June 1981, no. MIL-STD-721C.

[11] L. R. Harriott, "Limits of lithography," in *Proceedings of the IEEE*, 2001, vol. 89, no. 3, pp. 366–374.

[12] D. A. Buchanan, "Scaling the gate dielectric: materials, integration, and reliability," *IBM Journal of Research and Development*, vol. 43, no. 3, pp. 245–264, 1999.

[13] J.-H. Yoo, K.-C. Kim, K.-C. Lee, and K.-H. Kyung, "A 32-Bank 1 Gb self-strobing synchronous DRAM with 1 Gbyte/s bandwidth," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1635–1644, 1996.

[14] J. Bowles, "A survey of reliability-prediction procedures for microelecric devices," *IEEE Transactions on Reliability*, vol. 41, no. 1, pp. 2–12, 1992.

[15] H. H. Huston and C. P. Clarke, "Reliability defect detection and screening during processing-theory and implementation," in *Proceedings International Reliability Physics Symposium*, 1992, pp. 268–275.

[16] C. H. Stapper, "Modeling of defects in integrated circuit photolithographic patterns," *IBM Journal of Research and Development*, vol. 28, no. 4, pp. 461–475, 1984.

[17] B. T. Murphy, "Cost-size optima of monolithic integrated circuits," in *Proceedings of the IEEE*, 1964, vol. 52, pp. 1537–1545.

[18] R. B. Seeds, "Yield and cost analysis of bipolar LSI," in *1967 IEEE International Conference on Electron Devices Manufacturing*, 1967, p. 12.

[19] C. H. Stapper, "Defect density distribution for LSI yield calculations," *IEEE Transactions on Electron Devices*, vol. ED-20, pp. 655–657, 1973.

[20] T. S. Barnett, "Yield-reliability modeling for integrated circuits: Theory and experimental verification," Ph.D. dissertation, Auburn University, AL, 2002.

[21] T. Kim and W. Kuo, "Modeling manufacturing yield and reliability," *IEEE Transactions on Semiconductor Manufacturing*, vol. 12, no. 4, pp. 485–492, 1999.

[22] F. Kuper, J. Van der Pol, E. Ooms, T. Johnson, and R. Wijburg, "Relation between yield and reliability of integrated circuits: Experimental results and application to continuous early failure rate reduction programs," in *Proceedings International Reliability Physics Symposium*, 1996, pp. 17–21.

[23] J. Van der Pol, F. Kuper, and E. Ooms, "Relation between yield and reliability of integrated circuits and application to failure rate assesment and reduction in the one digit fit and ppm reliability era," *Microelectronics and Reliability*, vol. 36, pp. 1603–1610, 1996.

[24] T. S. Barnett and A. D. Singh, "Extending integrated-circuit yield-models to estimate early-life reliability," *IEEE Transactions on Reliability*, vol. 52, no. 3, pp. 296–300, 2003.

[25] *TOSHIBA Semiconductor Reliability Handbook*, Toshiba Co., Irvine, CA, 2001.

[26] S. Kikuda, H. Miyamoto, S. Mori, M. Niiro, and M. Yamada, "Optimized redundancy selection based on failure-related yield model for 64-Mb DRAM and

beyond," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 11, pp. 1550–1555, 1991.

[27] C. H. Stapper, J. A. Fifield, H. L. Kalter, and W. A. Klassen, "High-reliability fault-tolerant 16-Mbit memory chip," *IEEE Transactions on Reliability*, vol. 42, no. 4, pp. 596–603, 1993.

[28] C. H. Stapper and H. Lee, "Synergistic fault-tolerance for memory chips," *IEEE Transactions on Computers*, vol. 41, no. 9, pp. 1078–1087, 1992.

[29] C. H. Stapper, A. N. McLaren, and M. Dreckmann, "Yield model for productivity optimization of VLSI memory chips with redundancy and partially good product," *IBM Journal of Research and Development*, vol. 24, no. 3, pp. 398–409, 1980.

[30] C. H. Stapper, "Improved yield models for fault-tolerant memory chips," *IEEE Transactions on Computers*, vol. 42, no. 7, pp. 872–881, 1993.

[31] W. Kuo and M. J. Zuo, *Optimal Reliability Modeling: Principles and Applications*. Hoboken, NJ: John Wiley & Sons, 2002.

[32] R. E. Barlow and F. Proschan, *Mathematical Theory of Reliability*. Philadelphia, PA: SIAM, 1996.

[33] M. S. Chern, "On the computational complexity of reliability redundancy allocation in a series system," *Operations Research Letters*, vol. 11, no. 5, pp. 309–315, 1992.

[34] W. Kuo, V. R. Prasad, F. A. Tillman, and C. L. Hwang, *Optimal Reliability Design: Fundamentals and Applications*. Cambridge: Cambridge University Press, 2001.

[35] V. R. Prasad and W. Kuo, "Reliability optimization of coherent system," *IEEE Transactions on Reliability*, vol. 49, no. 3, pp. 323–330, 2000.

[36] W. Kuo and V. R. Prasad, "An annotated overview of system reliability optimization," *IEEE Transactions on Reliability*, vol. 49, no. 2, pp. 176–191, 2000.

[37] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. New York: John Wiley & Sons, 1988.

[38] K. B. Misra and U. Sharma, "An efficient algorithm to solve integer-programming problems arising in system reliability design," *IEEE Transactions on Reliability*, vol. 40, no. 1, pp. 81–91, 1991.

[39] Y. Nakagawa, K. Nakashima, and Y. Hattri, "Optimal reliability allocation by branch-and-bound technique," *IEEE Transactions on Reliability*, vol. 27, no. 1, pp. 31–38, 1978.

[40] E. L. Johnson, G. L. Nemhauser, and M. W. P. Savelsbergh, "Progress in linear programming-based algorithms for integer programming: An exposition," *INFORMS Journal on Computing*, vol. 12, no. 1, pp. 2–23, 2000.

[41] E. K. Lee and J. E. Mitchell, "Integer programming: Branch and bound methods," in *Encyclopedia of Optimization*, C. A. Floudas and P. M. Pardalos, Eds. Dordrecht: Kluwer Academic Publishers, 2001, vol. 2, pp. 509–519.

[42] X. L. Sun, K. I. M. McKinnon, and D. Li, "A convexification method for a class of global optimization problems with applications to reliability optimization," *Journal of Global Optimization*, vol. 21, pp. 185–199, 2001.

[43] M. Tawarmalani and N. V. Sahinidis, "Global optimization of mixed integer nonlinear programs: A theoretical and computational

study," 2003, submitted to Mathematical Programming, available at http://web.ics.purdue.edu/ mtawarma/.

[44] H. Tuy and L. T. Luc, "A new approach to optimization under monotonic constraint," *Journal of Global Optimization*, vol. 18, pp. 1–15, 2000.

[45] K. Gopal, K. K. Aggarwal, and J. S. Gupta, "An improved algorithm for relaibility optimization," *IEEE Transactions on Reliability*, vol. 29, pp. 325–328, 1978.

[46] J. H. Kim and B. J. Yum, "A heuristic method for solving redundancy optimization problems in complex systems," *IEEE Transactions on Reliability*, vol. 42, no. 4, pp. 572–578, 1993.

[47] D. W. Coit and A. Smith, "Adaptive penalty methods for genetic optimization of constrained combinatorial problems," *INFORMS Journal on Computing*, vol. 8, no. 2, pp. 173–182, 1996.

[48] ——, "Penalty guided genetic search for reliability design optimization," *Computers and Industrial Engineering*, vol. 30, no. 4, pp. 895–904, 1996.

[49] ——, "Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach," *Computers and Operations Research*, vol. 23, no. 6, pp. 515–526, 1996.

[50] Y. Nakagawa and S. Miyazaki, "Surrogate constraints algorithm for reliabilty optimization problem with two constraints," *IEEE Transactions on Reliability*, vol. R-30, no. 2, pp. 175–180, 1981.

[51] K. K. Aggarwal, "Redundancy optimization in general systems," *IEEE Transactions on Reliability*, vol. R-25, pp. 330–332, 1976.

[52] W. Kuo, C. L. Hwang, and F. A. Tillman, "A note on heuristic methods in optimal system reliability," *IEEE Transactions on Reliability*, vol. R-27, pp. 320–324, 1978.

[53] Y. Nakagawa and K. Nakashima, "A heuristic method for determining optimal reliability allocation," *IEEE Transactions on Reliability*, vol. R-26, no. 3, pp. 156–161, 1977.

[54] T. Kohda and K. Inoue, "A reliability optimization method for complex systems with the criterion of local optimality," *IEEE Transactions on Reliability*, vol. R-31, pp. 109–111, 1982.

[55] D. H. Shi, "A new heuristic algorithm for constrained redundancy-optimization in complex systems," *IEEE Transactions on Reliability*, vol. 36, no. 5, pp. 621–623, 1987.

[56] L. Jianping, "A bound heuristic algorithm for solving reliability redundancy optimization," *Microelectronics and Reliability*, vol. 36, no. 3, pp. 335–339, 1996.

VITA

Chunghun Ha received the B.S. degree in electronics engineering from Yonsei University, Seoul, Korea, in 1993, and the M.S. degree in industrial engineering from Texas A&M University in 2000. He received his Ph.D. degree in industrial engineering from Texas A&M University in 2004. Previously, he was a researcher at the Samsung Advanced Institute of Technology (SAIT). His research interests include reliability optimization, non-convex programming, integer and mixed-integer programming, reliability and yield analysis, modeling, and optimization in integrated circuits and micro/nano systems, and defect- and fault- tolerant architecture for the nanocomputer.

E-MAIL: chriver58@yahoo.com,

ADDRESS: Woosung APT 11-1203, Daechi-1-dong, Kangnam-Gu, Seoul, Korea